Story of Two GPUs: Characterizing the Resilience of Hopper H100 and Ampere A100 GPUs

Shengkun Cui*
University of Illinois Urbana-Champaign
Urbana, USA
scui8@illinois.edu

Archit Patke*
University of Illinois Urbana-Champaign
Urbana, USA
apatke@illinois.edu

Hung Nguyen
University of Illinois Urbana-Champaign
Urbana, USA
hungnt@illinois.edu

Aditya Ranjan
University of Illinois Urbana-Champaign
Urbana, USA
aranjan5@illinois.edu

Ziheng Chen
University of Illinois Urbana-Champaign
Urbana, USA
zihengc2@illinois.edu

Phuong Cao
University of Illinois Urbana-Champaign
Urbana, USA
pcao3@illinois.edu

Gregory Bauer
University of Illinois Urbana-Champaign
Urbana, USA
gbauer@illinois.edu

Brett Bode
University of Illinois Urbana-Champaign
Urbana, USA
brett@illinois.edu

Catello Di Martino
Nokia Bell Labs
Sao Paulo, Brazil
lelio.di_martino@nokia-bell-labs.com

Saurabh Jha
IBM Research
Yorktown Heights, USA
Saurabh.Jha@ibm.com

Chandra Narayanaswami
IBM Research
Yorktown Heights, USA
chandras@us.ibm.com

Daby Sow
IBM Research
Yorktown Heights, USA
sowdaby@us.ibm.com

Zbigniew T. Kalbarczyk
University of Illinois Urbana-Champaign
Urbana, USA
kalbarcz@illinois.edu

Ravishankar K. Iyer
University of Illinois Urbana-Champaign
Urbana, USA
rkiyer@illinois.edu

Abstract

This study characterizes GPU resilience in *Delta*[1], a large-scale AI system that consists of 1,056 A100 and H100 GPUs, with over 1,300 petaflops of peak throughput. We used 2.5 years of operational data (11.7 million GPU hours) on GPU errors. Our major findings include: (i) H100 GPU memory resilience is worse than A100 GPU memory, with 3.2x lower per-GPU MTBE for memory errors, (ii) The GPU memory error-recovery mechanisms on H100 GPUs are insufficient to handle the increased memory capacity, (iii) H100 GPUs demonstrate significantly improved GPU hardware resilience over A100 GPUs with respect to critical hardware components, (iv) GPU errors on both A100 and H100 GPUs frequently result in job failures due to the lack of robust recovery mechanisms at the application level, and (v) We project the impact of GPU node availability on larger-scales and find that significant overprovisioning of 5% is necessary to handle GPU failures.

CCS Concepts

• **Computer systems organization** → **Reliability**; **Availability**; **Redundancy**; • **Computing methodologies** → **Discrete-event simulation**; • **General and reference** → *Measurement*; *Reliability*.

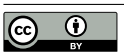
Keywords

Large-scale AI/HPC System; Reliability Evaluation and Analysis; GPU Resilience; Application Impacts.

ACM Reference Format:
Shengkun Cui, Archit Patke, Hung Nguyen, Aditya Ranjan, Ziheng Chen, Phuong Cao, Gregory Bauer, Brett Bode, Catello Di Martino, Saurabh Jha, Chandra Narayanaswami, Daby Sow, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2025. Story of Two GPUs: Characterizing the Resilience of Hopper H100 and Ampere A100 GPUs. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3712285.3759821

*Equal contribution.

[1] *Delta* is an HPC system operated by the National Center for Supercomputing Applications (NCSA) at the University of Illinois Urbana-Champaign.



SC '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1466-5/25/11
https://doi.org/10.1145/3712285.3759821

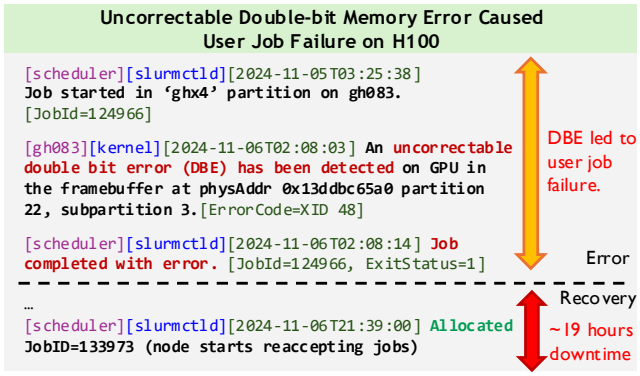


Figure 1: A double-bit memory error (XID 48) occurred, and it is uncorrectable by the SECDED ECC HBM3 memory. Due to this double-bit error, the user job scheduled on that GPU failed, as reflected in the scheduler logs. Subsequently, this uncorrectable memory error requires a node draining and reset to complete the row remapping recovery action. The total recovery process for this incident took 19 hours, during which the node was unavailable for accepting new jobs. This incident shows that a GPU error can lead to user job failures and significantly impact node availability.

1 Introduction

Large-scale HPC systems are important not only for scientific workloads [49] but also for data analytics [1] and machine learning (ML) [22]. The main components of these systems are specialized accelerators, such as GPUs, that enable acceleration of computations, such as ML training [2, 53], ML inference [26], and simulations [41, 48].

This paper studies the resilience of A100 GPUs and compares the result with H100 GPUs of the GH200 Grace Hopper Superchips[2], together with their associated memory: 40 GB HBM2e on each A100 GPU, and 96 GB HBM3 on each H100 GPU, respectively. The A100 GPU nodes and H100 GPUs (GH200 Superchip) nodes are operated as two independent systems sharing a storage cluster running the Lustre file system, allowing us to study and compare the two systems (refer to as *Delta*). The workflow on *Delta* [6] involves users from universities nationwide and presents a spectrum of HPC and ML workloads. The study uses 2.5 years of data on critical GPU errors collected across the stated GPUs, encompassing 9.6 million GPU hours of A100 GPUs and 2.1 million GPU hours of H100, a combined 11.7 million GPU hours.

This study assesses (i) the resilience of GPU hardware and memory components; (ii) the error propagation paths in GPU memory, GPU hardware, and NVLink interconnect; and (iii) the impact of the observed GPU errors on user jobs. Figure 1 shows an example error propagation path for an uncorrectable double-bit memory error in H100 GPU that caused user job failure. The complete recovery process required node draining and GPU reset, which took 19 hours following the error detection.

Our major findings include:

(i) H100 shows 3.2× lower per-GPU mean time between errors (MTBE) compared to A100 for uncorrectable ECC memory errors. The per-GB MTBE of the H100's HBM3 memory is 24% lower (~ 8.5M hours) than the A100's HBM2e memory (~ 11.3M hours). We conjecture that the reduction in memory resilience stems from H100's higher memory capacity.

(ii) The GPU memory error-recovery mechanisms on A100 and H100 GPUs (e.g., memory row remapping, error containment) [37] improve GPU memory resilience and reduce service interruption. We observed that these mechanisms mitigate (e.g., using memory row remapping) 92% of uncorrectable ECC memory errors on H100 GPUs. However, the memory error-recovery mechanism is insufficient to handle the increase in memory capacity and the corresponding increase in row remapping events on the H100 GPUs.

(iii) H100 GPUs demonstrate significantly improved hardware resilience over A100 GPUs, with respect to critical components such as GSP[3], NVLink, and PMU SPI[4], which were major sources of job failures in A100 systems. We attribute this to driver-level enhancements and tighter integration [34, 36], which contribute to improved resilience. Specifically, comparing H100 and A100 GPU hardware, we observed (a) a significant reduction of GSP errors on H100 (only 3 cases in our measurement period) and (b) the elimination of PMU SPI error propagations, which on A100 GPUs can lead to MMU errors 88% of times with 90% leading to user job failures, (c) no NVLink errors on H100 GPUs during the measurement period.

(iv) GPU errors on both A100 and H100 GPUs frequently result in job failures due to the lack of robust recovery mechanisms at the application level. Except for MMU and NVLink errors, other GPU errors cannot be handled by application-level mechanisms, resulting in close to 100% job failure rate. The underlying cause of job failures differs by GPU type: hardware errors are the predominant cause in A100 GPUs, whereas memory errors are the primary cause in H100 GPUs. Overall, we find that application-based recovery strategies are largely ineffective; hence, there is a compelling need to improve resilience at the GPU memory and hardware level.

(v) The overall availability per-GPU node is approximately 99.4% for A100 GPUs and 99.3% for H100 GPUs, corresponding to a downtime between 9–10 minutes per day. We projected the impact of this measured availability on larger scales via emulation. For example, to maintain 99.9% availability at the job level, overprovisioning of 5% would be necessary. While at first glance, such overprovisioning would appear to be a small cost, for the above example, it would cost over $1 million per month. If GPU node availability were improved to 99.9%, the required overprovisioning would reduce by 2.5×.

Putting the paper in perspective. Previous studies on characterizing GPU resilience in large-scale systems [11, 12, 18, 19, 32, 33, 38, 39, 51, 52] focus on GPU memory errors in older GPU generations (Tesla, Kepler, and Volta) that lack the latest resilience (e.g., row remapping, error containment, NVLink CRC-retry) and performance features (e.g., GPU System Processor) introduced in NVIDIA Ampere-generation GPUs. A recent study from Meta [25] characterizes cluster-level resilience for two large-scale machine learning clusters equipped with A100 GPUs. Our paper provides a deeper understanding of GPU errors and failures of two recent

[2] GH200 Superchip, hereafter.

[3] A GPU system processor (GSP) is an onboard co-processor that offloads driver tasks from the CPU for latency and performance improvement.

[4] PMU on an NVIDIA's GPU regulates the frequency, voltage, and power of the GPU based on various factors such as temperature and power cap. SPI stands for serial peripheral interface, which serves as the communication channel between peripheral hardware.

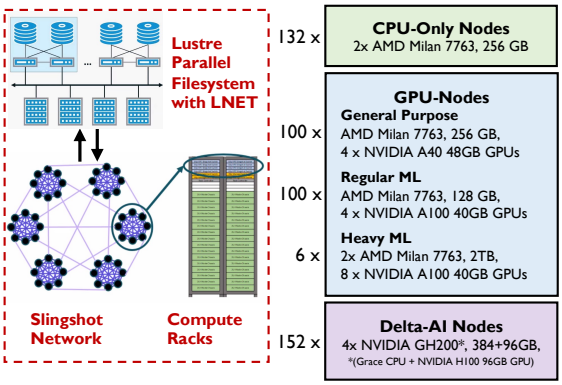


Figure 2: System architecture and specifications of *Delta*. This study focuses on the H100 and A100 GPU nodes.

generations of GPUs, A100 and H100, and their impact on a broad set of HPC/ML applications. To the best of our knowledge, this is the first study on A100 and H100 GPU errors in HPC/ML systems.

All software, datasets, and analysis scripts are available in the AD/AE materials and at https://doi.org/10.5281/zenodo.15287639.

2 Background

This section provides information on (i) *Delta* specification, (ii) *Delta* overall GPU utilization, GPU operational environment, and workloads, (iii) critical GPU error categories used in this study, and (iv) GPU error management and recovery.

2.1 *Delta* Specifications

***Delta* Specifications.** Figure 2 shows the layout of *Delta*, which consists of a cluster of 106 4- and 8-way A100 40 GB GPU nodes with 448 A100 GPUs, and in addition, it has a second cluster of 152 4-way GH200 nodes with 608 H100 96 GB GPUs, a total of 1,056 A100 and H100 GPUs. The two clusters share a common storage cluster running Lustre Filesystem. Our study focuses on the resilience of A100 GPUs and H100 GPUs (the GPU of GH200), as they are optimized for AI/ML workloads, exhibit the highest utilization, and incorporate the latest resilience features.

Note that the H100 GPUs studied here are integrated in GH200 Superchips, tightly coupled to NVIDIA Grace CPUs via NVLink-C2C interconnect, distinguishing them from discrete H100 GPUs. While the H100 microarchitecture and memory specifications are consistent across both variants, differences in CPU–GPU integration may lead to variations in resilience characteristics.

2.2 *Delta* GPU Operational Environment

A100 and H100 GPUs ran in comparable operational conditions in terms of (i) utilization, (ii) cooling/temperature, and (iii) workloads.

Overall Utilization. *Delta*'s NVIDIA A100 GPUs are frequently scheduled and utilized, with an average GPU utilization of 51% during the operational period. NVIDIA H100 GPUs show a slightly lower average utilization (41%) than A100 GPUs.

Cooling and Temperature. *Delta*'s A100 and H100 GPUs are liquid-cooled from shared facility water supply through independent cooling loops. At average utilizations, their mean temperatures are 40°C (A100) and 37°C (H100), with maximums of 49°C and 48°C, respectively, indicating comparable operational conditions.

Workloads. We use the allocated *Delta* projects' Field-of-Science distributions as a proxy for workload characterization. Both GPUs handled similarly diverse workloads with comparable distributions across the top five fields (A100s/H100s): Computer Science (30.4/32.8%), AI & Intelligent Systems (18.1/29.1%), Applied CS (4.7/4.2%), Biophysics (5.1/1.7%), Materials Engineering (3.6/1.4%) for A100/H100, respectively.

2.3 NVIDIA GPU Error Categories

NVIDIA GPU errors are reported as XID errors. In this study, we selected a subset of XID errors that are described as common and high-impact by NVIDIA's Developer Manuals [35, 37], NVIDIA Developer Forums and Blogs, and *Delta* site reliability engineers (SREs). We primarily collected errors and their associated recovery events. The selected XID errors/events indicate GPU issues that often cannot be resolved without SRE's interventions (e.g., node service and GPU replacement). The selected XIDs and their corresponding GPU errors are described in Section 4, Table 1. We categorize the selected GPU errors into three categories: (i) GPU hardware, which includes all onboard hardware except for GPU memory and NVLink interconnect, (ii) NVLink interconnect, and (iii) GPU memory. Note that General GPU Software Error (XID 13) and Reset Channel Verification Error (XID 43) are usually caused by user jobs and do not impact the health of the GPU [35]; we excluded those errors from our study.

GPU Hardware Errors. The critical GPU hardware errors we studied include MMU[5] errors, GPU Fallen Off the Bus errors, GSP errors, and PMU communication errors. We do not consider other GPU hardware errors in our study. GPU hardware errors can lead to user job failures, GPU halt, and data corruption. Among those errors, GPU Fallen Off the Bus and GSP errors lead to GPU failures, and manual GPU resets or node reboots are required to recover from the error [35]. *Delta* SREs monitor GSP errors closely to ensure timely recovery to maintain GPU availability.

GPU Interconnect (NVLink) Errors. GPU-GPU NVLink errors are caused by faulty GPU hardware, connectors, or improper connector installation during system integration, and can lead to GPU unavailability and user job failures. NVLink errors impede data transfer between GPUs and reduce computational throughput. A GPU reset or node reboot is required to clear NVLink errors [35].

GPU Memory Errors. GPU memory errors included in this study are double-bit errors (DBEs) and consecutive single-bit errors (SBEs)[6]. Individual SBEs are not logged, as they are automatically corrected by ECC. DBEs and consecutive SBEs are considered uncorrectable ECC memory errors by the NVIDIA driver, and they trigger downstream error-recovery mechanisms [35, 37], which are introduced in Section 2.3. Failures in these mechanisms can lead to GPU/node failures and require GPU or node reboots to recover [35]. *Delta* SREs continuously monitor uncorrectable ECC memory errors and error-recovery failures to ensure timely replacement of faulty GPUs.

[5]The memory management unit (MMU) provides essential memory I/O functionalities.
[6]Consecutive SBEs are multiple single-bit error (SBE) occurrences at the same memory location.

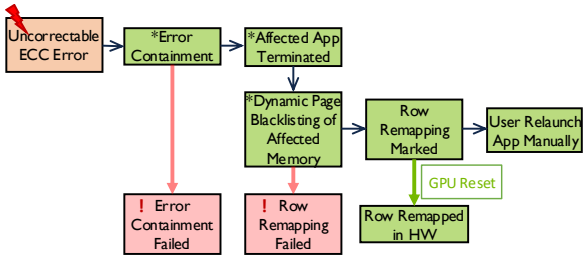


Figure 3: NVIDIA memory error recovery process for A100 and H100 GPUs.

2.4 NVIDIA GPU Error Management

Here, we provide an overview of the resilience architecture of NVIDIA A100 and H100 GPUs.

GPU Memory. Figure 3 shows the uncorrectable ECC memory error-recovery process [37] for A100 and H100 in more detail. The primary mechanism for mitigating uncorrectable ECC memory errors for A100 and H100 GPUs is *row-remapping*, wherein the faulty memory row is replaced with a spare row, and a row-remapping event (RRE) is logged. The actual row remapping happens at the next GPU reset (e.g., during node reboot or maintenance). If there are no spare memory rows, a row remapping failure (RRF) is indicated [35, 37].

A100 and H100 GPUs support online recovery mechanisms such as *error containment* and *dynamic page offlining* [35, 37] for mitigating uncorrectable ECC memory errors with minimal node interruption. The dynamic page offlining marks the faulty memory page as unusable without requiring a GPU reset to maintain availability. The error containment procedure terminates user processes using the faulty memory address to prevent error propagation to other applications. Successful error containment is logged as a Contained Memory Error, whereas an unsuccessful error containment is logged as an Uncontained Memory Error. Failure in a row-remapping or error containment can cause a GPU failure that requires a GPU reset or node reboot. *Delta* SREs monitor row-remapping failures and replace GPUs that repeatedly emit such errors.

GPU Hardware. While the GPU caches and memory are SECDED protected, information on failure-recovery mechanisms on GPU hardware, including peripheral hardware such as GSP, PMU, or SPI communication channels, is limited.

GPU Interconnect (NVLink). NVLink employs Cyclic Redundancy Checks (CRCs) for error detection to ensure the integrity of flow control digits and data. Upon encountering a CRC checksum error, NVLink retries packet transmissions from the last-known good packet.

3 Methodology

3.1 Data Sources

Our analysis was performed on data collected from *Delta* over its operational period: (i) 895 days from October 2022 to March 2025 for A100 GPUs and (ii) 146 days from October 2024 to March 2025 for H100 GPUs, covering 11.7 million GPU hours. This section describes data sources for *Stage I*: data collection and extraction in the pipeline in Figure 4.

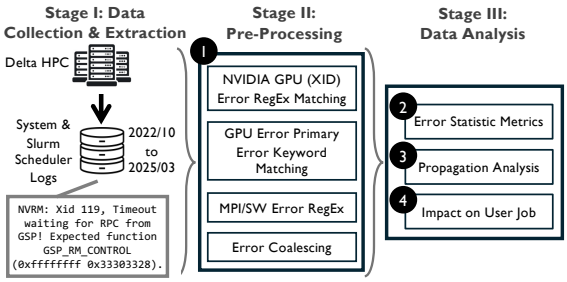


Figure 4: Overview of our data collection, processing, and analysis pipeline.

Algorithm 1: Error Coalescing and Persistence Analysis.

Input : Error logs with timestamps $E = \{(e_1, t_1), \ldots, (e_n, t_n)\}$, regex patterns $\mathcal{R} = \{r_1, r_2, \ldots\}$, time window Δt
Output: Coalesced errors with persistence duration E'

```
E' ← ∅ // Initialize output set
foreach pattern r ∈ R do
    E_r ← {(e_i, t_i) ∈ E | e_i matches r} Filter errors with regex
    i ← 1
    // Loop through errors in a matched group
    while i ≤ |E_r| do
        (e_first, t_start), t_latest ← (e_i, t_i), t_i
        // Loop through later errors within the matched group while
        i + 1 ≤ |E_r| do
            (e_next, t_next) ← (e_{i+1}, t_{i+1})
            // Error has identical message and is close in time if
            e_next = e_first and t_next − t_latest ≤ Δt then
                t_latest ← t_next //Discard latest error
                i ← i + 1
            else
                break
        // Store coalesced error and persistence duration
        Add (e_first, t_start, t_latest − t_start) to E'
        i ← i + 1 // Move to the next unprocessed error
return E'
```

System logs. System logs collected from all compute nodes capture events across system components. We created a set of regular expression (RegEX) patterns and used it to extract GPU error-recovery log entries by referring to NVIDIA XID messages [35] from the system logs (Figure 4, item (1)). The GPU error logs were our major sources of error and recovery information.

Slurm scheduler database. *Delta* uses the Slurm Workload Manager [60] ("Slurm scheduler" hereafter) for scheduling user jobs. The Slurm scheduler database keeps track of user job scheduling events, including the start and end times, the scheduled nodes, resource usage, job status, exit codes, and the srun command line. We used the Slurm database for user job failure characterization.

NVIDIA DCGM database. *Delta* uses NVIDIA Data Center GPU Manager (DCGM) to collect metrics and status data from all GPUs with a one-minute granularity. We used DCGM metric data to characterize GPU utilization.

3.2 Data Processing Pipeline

This section focuses on *Stage II and III* of the data processing pipeline in Figure 4, which pre-processes the raw logs, compute error counts, mean times between errors (MTBE), error propagation, and impact on user jobs.

Error Coalescing Analysis. The error coalescing step in Figure 4, item (1) filters out duplicated errors. While most errors are logged as isolated events, there are frequent periods during which

the same error is logged repeatedly in close succession, i.e., there are error bursts. During these bursts, the system continually detects and attempts to recover from errors, which could lead to either system recovery or failure. To prevent over-counting, we assume that identical error logs within a short time interval (Δt) from the same GPU are caused by the same issue. Thus, the error coalescing step counts only the first occurrence by combining identical error log lines from the same GPU within a predefined time interval (Δt) into a single error (see Algorithm 1). The remaining analyses in this paper were conducted on errors after the coalescing. We set $\Delta t = 5$ as decreases in Δ results in more duplicated logs, while further increase in Δ results in negligible changes in coalesced log count.

Error Statistic Metrics. Using the coalesced error logs as input, the pipeline computes standard error statistics metrics including the error count and the MTBE as in [38] (Figure 4, item (2)). MTBE allows fair comparison between GPU types with different total operational hours. For errors that are not directly countable from XID logs, we estimate their occurrence by correlating related XIDs within the same time interval [35, 37, 38]. In particular, the number of uncorrectable ECC memory errors can be inferred by summing up the number of RREs and RRFs, as they are mutually exclusive outcomes of an uncorrectable ECC memory error recovery event. Subsequently, the number of consecutive SBEs can be obtained by subtracting the number of DBEs from the number of uncorrectable ECC memory errors.

We additionally computed system-wide MTBE and derived per-node MTBE by normalizing the error count using the number of GPU nodes in *Delta*. The per-node MTBE indicates the operational hours a single *Delta* GPU node can function before encountering an error. For GPU memory resilience characterization, we additionally derived per-GPU MTBE by normalizing the error count using the number of GPU in a node and per-GB MTBE by normalizing the error count with per GPU memory capacity in GB. The per-GPU MTBE reflects the operational hours a single GPU can function before encountering an error.

Error Propagation Analysis. We performed error propagation analysis (Figure 4, item (3)) to capture how errors propagate within a GPU and across different GPUs while measuring the propagation time. The propagation probability from GPU error e_1 to e_2 is defined as $P(e_2|e_1) = \frac{\#e_2}{\text{Total } \#e_1}, \quad t_{e_2} - t_{e_1} \leq \Delta t$. A propagation path is created if e_2 occurs immediately after e_1 within a predefined time window Δt. If there is no succeeding error e_2 after e_1 within Δt, then e_1 is a terminal error that does not propagate. For intra-GPU[7] propagation, we require errors e_1 and e_2 to be on the same GPU device, whereas for the inter-GPU propagation, e_1 and e_2 are from two distinct GPUs on the same node. We recorded the time difference between the initial (e_1) and subsequent errors (e_2), referred to as the *propagation time*, for each propagation event. A shorter propagation time suggests a higher correlation between e_1 and e_2. We applied the same Δt selection criteria as the error coalescing analysis and selected the $\Delta t = 5$ seconds in error propagation analysis.

User Job Impact Analysis. The user job impact analysis step (Figure 4, item (4)) associates GPU errors with failed user jobs to characterize the impact of GPU errors on user jobs. Section 5 provides detail on this analysis.

[7]GPU devices are identified by their node ID and PCI Express bus address.

4 Characterizing GPU Resilience

This section characterizes and compares the resilience of *Delta*'s NVIDIA A100 (Ampere) and H100 (Hopper) GPUs. Specifically, we discuss error statistics and error propagation of GPU errors in three categories: (i) GPU memory, (ii) GPU hardware, and (iii) NVLink interconnect, as described in Section 2.3 and Table 1. These errors are critical because they interrupt user jobs and lead to unplanned node downtime, as we show in Section 5. We first highlight error statistics and key findings from our analysis and then focus on error propagation for each of the three error categories on *Delta*'s workload. Note that we do not directly compare *Delta* with *Blue Waters*[12], *Titan*[32], or *Summit* [38] as those systems used older GPUs lacking latest performance and resilience features central to our study, e.g., GSP, dynamic page offlining, row remapping, memory error containment, and NVLink recovery.

4.1 Error Statistics and Result Highlights

Table 1 summarizes the selected critical GPU error statistics for A100 GPUs and H100 GPUs during the operational period, including error count, system-wide mean time between errors (MTBE) and per-node MTBE. As described in Section 2.3, these selected errors are critical XID errors that can lead to user job interruption and node downtime, requiring manual SRE intervention for recovery. In the operational period, a total of 14,821 critical errors (listed in Table 1) were recorded for *Delta*'s A100 GPUs, with a system-wide MTBE of 1.4 hours and node MTBE of 154 hours. *Delta*'s H100 GPUs encountered 1,821 GPU errors, with a system-wide MTBE of 1.9 hours and node MTBE of 292 hours, higher than A100 nodes. Next, we use the results from Table 1 to assess the resilience characteristics of GPU memory, GPU hardware, and NVLink interconnect in greater detail.

GPU Memory. Recall that, each A100 GPU incorporates 40 GB of HBM2e memory, and each H100 GPU incorporates 96 GB of HBM3 memory. H100's shows 3.2× lower per GPU MTBE for uncorrectable ECC memory errors compared to A100's, despite comparable per-GB MTBE. This reduced resilience likely stems from H100's higher memory density, which trades resilience for capacity and performance. The observed lower per node MTBE of RREs and higher RRF counts on H100 GPUs further suggest that current resilience features are insufficient for increased memory capacity. As a uncorrectable memory error can cause a multi-GPU job to fail, enabling uninterrupted execution in such cases at scale remains an open challenge. We present the detailed analysis below.

(i) *Delta*'s H100 GPUs exhibit lower memory resilience than the A100 GPUs. Specifically, the per GPU MTBE is 3.2× lower on H100 GPUs at 88,768 hours versus 283,271 hours for A100 GPUs. Based on the per GPU MTBE, we calculated the per GB MTBE to be 8,521,728 hours[8] for H100's HBM3 memory vs. 11,330,826 hours for A100's HBM2e memory, a 24% reduction.

[8]We calculated the per GB MTBE by multiplying the per node MTBE with the total memory of all GPUs in GB for that node. For example, in a 4-way H100 GPU node with 96 GB of memory, the per GB MTBE = $22,192 \times 96 \times 4 = 8,521,728$ hours.

Table 1: *Delta* NVIDIA Ampere A100 and Hopper H100 (GPU of GH200 Superchip) GPU resilience statistics.

Event Code	Abbr.	Category	Description	Recovery Action	Count		MTBE (hrs)			
					A100	H100	System-wide		Per Node	
							A100	H100	A100	H100
XID 31	MMU Error	Hardware	GPU memory management unit (MMU) error.	MMU error due to invalid memory access or driver/hardware bugs.	8,863	1,737	2.4	2	257	307
XID 48	DBE	Memory	Double bit ECC memory error (DBE).	Triggers RRE; GPU reset or node reboot is needed to clear this error.	1	17	–	206	–	31,330
–	Consecutive SBEs	Memory	Consecutive single-bit ECC memory errors (SBEs).	Triggers RRE; GPU reset or node reboot is needed to clear this error.	33	7	651	501	68,996	76,087
–	Uncorrectable ECC memory Errors	Memory	Consecutive SBEs or a DBE.	Triggers RRE; GPU reset or node reboot is needed to clear this error.	34	24	632	146	66,967	22,192
XID 63	RRE	Memory	Row remapping event triggered by uncorrectable MBE: one DBE or two SBEs at the same memory address.	GPU reset needed for row remapping.	34	16	632	219	66,967	33,288
XID 64	RRF	Memory	Row remapping failure of a row remapping event.	GPU reset is needed to clear this error.	0	8	–	438	–	66,576
XID 74	NVLink Error	Inter-connect	NVLink error indicating connection issues between GPUs via NVLink interconnection.	GPU reset or SRE intervention required.	1,922	–	11	–	1,185	–
XID 79	GPU Fallen Off the Bus Error	Hardware	GPU has fallen off the system bus and is not reachable, typically because of driver or hardware issues.	GPU reset or SRE intervention required.	10	–	2,148	–	227,668	–
XID 94	Contained Memory Error	Memory	Uncorrectable contained ECC error, indicating successful containment.	Not specified.	13	14	1,652	250	175,144	38,043
XID 95	Uncontained Memory Error	Memory	Uncontained memory error, indicating failure in containment.	GPU reset or SRE intervention required.	11	19	1,953	184	206,989	28,032
XID 119/120	GSP Error	Hardware	NVIDIA GPU Systems Processor (GSP) error.	GPU reset or SRE intervention required.	3,857	3	6	1,168	590	177,536
XID 122/123	PMU SPI Error	Hardware	PMU SPI read/write failure, indicating failed communication with the PMU.	Not specified.	77	–	279.0	–	29,570	–

*NVIDIA A100/H100 GPU supports page retirement and up to 512 row remappings (RRE); previous generations support only 64 page retirements (no row remapping support).
*Row remapping, contained memory error, and uncontained memory error are new resilience features introduced starting with the NVIDIA Ampere for managing uncorrectable memory errors.
*Per-node MTBE (hrs) is derived by multiplying system MTBE by the number of GPU nodes of that GPU type. The number of nodes and GPUs per node type are specified in Figure 2.
*All XID events presented, except for row remapping events, are errors. However, for simplicity, we treat all XID events as errors in this paper.
*Uncorrectable ECC and Consecutive SBEs are inferred from the corresponding recovery event RRE and RRF (see Section 3.2).

We attribute the decrease in resilience is primarily due to the higher memory capacity (96 GB vs. 40 GB, a 2.4× increase), which increases the chances of bit flips. We additionally hypothesize that H100 memory resilience is worse due to (a) a lower signaling voltage that increases susceptibility to bit flips [58] and (b) an increased number of stacks that make heat dissipation challenging and degrade the resilience of memory modules, of the HBM3 memory.

(ii) Uncorrectable ECC memory error-recovery mechanisms (e.g., row remapping and error containment, see Section 2.3) improve GPU memory resilience and reduce service interruptions [37]. The error recovery mechanisms mitigate uncorrectable memory errors with a probability of 0.92 on H100 according to error propagation analysis in Section 4.2. However, these mechanisms may not scale well with the increase in GPU memory capacity in H100. For example, the available spare rows for row remapping are capped at the same 512 rows [37], which is not proportional to the 2.4× increase in memory capacity. Such insufficiency can be evident from the significantly lower per node MTBE of RRE on the H100 GPUs than the A100 GPUs, indicating more frequent recovery events. In addition, we observed 8 RRFs on H100 GPUs during the early operational period, which indicates memory recovery failure due to exhaustion of spared memory rows. We have yet to observe an RRF on the A100 GPUs during the much longer operational period.

GPU Hardware and NVLink Interconnect. GPU hardware such as GSP, PMU SPI, and NVLink are critical components from a resilience perspective for A100 GPUs, leading to node downtime and job failures. In this context, H100 GPUs demonstrate significant improvements in GPU hardware resilience, especially in critical components such as the GSP, PMU SPI, and NVLink. We believe that these improvements are likely due to the tightly integrated heterogeneous CPU-GPU architecture of GH200 [36] and driver-level enhancements [34]. We present a detail analysis below.

(i) Among the GPU hardware components on A100 GPUs, GSP, intended as a performance enhancement, is the most vulnerable due to its lack of robust detection and recovery. Our error propagation analysis (Section 4.2) shows that over 99% of GSP errors put the GPU in an error state and lead to job failure if encountered. A GSP error requires a GPU reset or a node reboot to recover, introducing significant overheads. In addition, PMU SPI communication errors propagate downstream and cause MMU errors with a probability of 0.88, leading to MMU errors, which then, in turn, result in a job failure over 90% of the time. Although PMU SPI communication errors are high-impact errors, they are not highlighted in NVIDIA's developer's manual [35].

(ii) Despite error detection mechanisms such as CRC and recovery mechanisms such as message retransmitting, NVLink GPU

interconnect errors are frequent (1,922 total NVLink errors) on A100 GPUs, with a node MTBE of 1,185 hours (system-wide MTBE of 11 hours); 42% (801) of the NVLink errors affected two or more GPUs. Although NVIDIA [35] indicates that a GPU reset or a node reboot is required to clear NVLink, *Delta* SREs suggested that NVLink errors were largely benign and did not always lead to job failures. Indeed, the user job impact analysis shows that an NVLink error has a 54% chance of leading to a job failure (see Section 5). When an NVLink error is observed but does not affect a user job, it may be because NVLink is primarily used for communication rather than computation in many jobs or because multiple NVLink errors within the same job have a consolidated impact, resulting in high occurrence rates but minimal application disruption.

(iii) H100 GPUs have substantially improved GPU hardware resilience over A100 GPUs: only 3 GSP errors were observed during the measurement period. Moreover, the number of GPU Fallen off the bus errors, NVLink errors[9], and PMU SPI errors were not observed in H100 GPUs. We attribute this improvement to (a) the better packaging and tight of CPU and GPU into a single heterogeneous compute module, which significantly reduces integration errors and enhances the resilience of the CPU-GPU complex, and (b) the NVIDIA driver upgrades appear to have improved GSP and NVLink stability (also observed by the *Delta* SREs).

GPU Errors Temporal Analysis. Temporal analysis revealed that GPU error rates followed the classic "bathtub" reliability curve. In the pre-operational period, i.e., the "infant-mortality" phase, *Delta* underwent extensive testing, hardware replacements, and software fixes, during which the system-wide MTBE was 0.15 hours[10]. In the operational period, the system-wide MTBE rose 10× to 1.4 hours, marking a transition to the "normal-life phase" with lowered error rate. Furthermore, estimation of the hazard-rate ($H(t) = \int h(t)\,dt$) and cumulative hazard-rate using Nelson-Aalen estimator showed that GPU hazard-rate varies over time but exhibits no clear temporal trends during the operational period (e.g., hazard rate with mean 0.7, std 7.94 for A100 GPUs).

GPU Failure and Replacement. During the measurement period (895 days) of A100 GPUs, four A100 GPUs were replaced due to GPU's failure to boot-up. In comparison, during the measurement period (146 days) of the H100 GPUs, two GPUs were replaced due to uncorrectable memory errors and row remapping failures. This provides an additional support for the observed result that memory is a resilience weak link on H100 GPUs, while the A100 GPUs are more susceptible to errors from GPU hardware components.

4.2 GPU Error Propagation

This section describes results on both *intra-GPU* and *inter-GPU error propagation* during the operational period for *Delta*'s A100 and H100 GPUs. Understanding GPU error propagation reveals resilience weak links between GPU components. We break down the error-recovery propagation into three categories for the GPU errors listed in Table 1: (i) GPU memory, (ii) GPU hardware, and (iii) NVLink interconnect, and estimate the propagation probabilities from GPU error logs (see Section 3.2). The propagation paths in Figures 5 to 7 are highlighted with lightning signs that indicate the beginning of each path. If two succeeding errors occur in close successions, i.e., a short propagation time, it suggests causality.

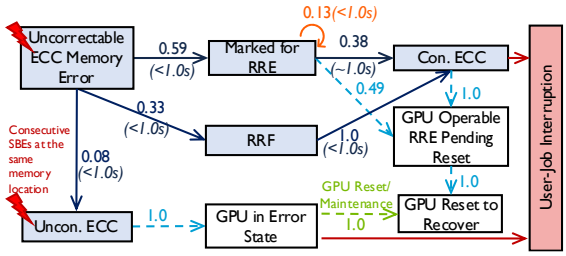


Figure 5: Intra-GPU uncorrectable memory error recovery paths in H100 GPUs. Numbers on the edges show propagation probability. The precise sub-second timing information is not available in the H100 nodes' system logs. All propagation time for H100 GPUs memory errors were within one second.

For GPU memory errors, we observed propagation primarily on H100 GPUs, which coincides with our findings on worsened memory resilience in Section 4.1. For GPU hardware and NVLink interconnect errors, we observed error propagation paths primarily on A100 GPUs because of the improved hardware resilience of H100 as discussed in Section 4.1.

4.2.1 ***GPU Memory-Error Propagation.*** Intra-GPU uncorrectable memory error recovery paths are shown in Figure 5 for H100 GPUs during the operational period. Memory error recovery path for A100 GPUs were a subset shown in Figure 5. Hence, we only show the memory propagation path for H100 GPUs.

Successful Error Recovery and Containment. As shown in Figure 5, row remapping recovery (RRE) triggered by an uncorrectable ECC memory error has a success rate of 0.59 on H100 GPUs. For the 33% of the row remapping events that fail (RRF), the GPU still contains the uncorrectable memory error because only the affected user jobs were terminated, and only the faulty page was offlined. Overall, considering both uncorrectable memory error recovery paths (RRE and error containment after an RRF), the impact of uncorrectable memory errors was alleviated 92% of the time on H100, while the GPU can remained operable. Such uninterrupted operations were not achievable on previous-generation GPUs (e.g., Kepler in [12, 51] and Volta in [38]), as an uncorrectable memory error would immediately cause user job interruption and put GPU in an error state, necessitating a GPU reset to recover [35].

Unsuccessful Error Containment. The above uncorrectable ECC memory error recovery or containment process can fail, resulting in uncontained memory errors (Section 2.3). The error containment process failed 8% of the times on H100 GPUs during the operational period (see Figure 5). Moreover, as informed by the *Delta* SREs, uncontained memory errors can be highly bursty and persistent and may spam the console logs, consuming useful compute cycles and leaving the GPU inoperable. We did not observed highly bursty and persistent uncontained memory error in A100 or H100 GPUs during the operational period.

Overall, the propagation analysis suggests that the new memory error recovery mechanisms on H100 GPUs alleviated impact of uncorrectable memory errors 92% of the time. That said, *Delta*'s H100 GPUs exhibited significantly more memory error recovery

[9]We tested the NVLink to ensure that the NVLinks were enabled.
[10]Evaluated using the critical-errors listed in Table 1.

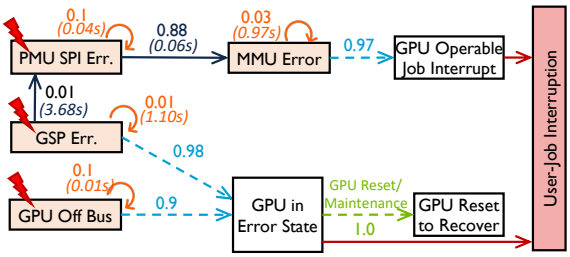


Figure 6: Intra-GPU hardware error propagation probabilities in A100 GPUs. Numbers on the edges show propagation probabilities and average propagation time in seconds.

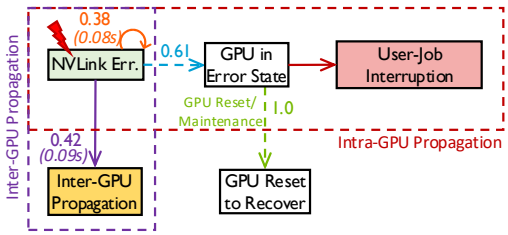


Figure 7: NVLink intra-GPU and inter-GPU error propagation in A100 GPUs. Numbers on the edges show propagation probabilities and average propagation time in seconds.

propagations than A100 GPUs, highlighting the worsened memory resilience compared to A100s. In addition, the highly bursty and persisting nature of uncontained memory errors can lead to node/system operation disruptions, as we learned from *Delta* SREs.

4.2.2 ***GPU Hardware-Error Propagation.*** This section primarily focuses on A100 because the H100 hardware (these components) experienced almost no error during the operational period. Figure 6 shows error propagation across GPU hardware components in A100 GPUs during the operational period. We found three dominant GPU hardware error propagation paths, originating in (i) GSP (GPU System Processor) errors, (ii) PMU (Performance Management Unit) SPI errors, and (iii) GPU Fallen Off the Bus errors. We omit hardware error propagation graphs for H100 GPUs as we only observed three GSP errors and their propagation paths are similar to that of the A100 GPUs.

GSP-related Errors. Error propagations that originate in GSP-related errors are the most prominent among GPU hardware errors (see Table 1) on A100 GPUs. A GSP error arises when the GSP fails to respond to the remote procedure calls from the GPU driver. Figure 6 shows that, with a probability of 0.99, GSP errors lead to the recurrence of the same error or put the GPU in an inoperable state. The remaining 0.01 (15 cases) of GSP errors caused PMU SPI communication errors (see the follow-up description) that led to user job failure, as depicted in Section 5. Our analysis additionally shows that 99% of GSP errors appeared in isolation without a preceding error.

GSP errors can be caused by either GSP firmware bugs [34] or demanding workload. For example, *Delta* SREs observed that these errors were highly correlated with heavy ML benchmarks, and they suggested that GSP errors are high-impact errors whose recovery requires manual node draining and reboots. Our propagation analysis confirm that the GSP is a single point of failure on both A100 GPUs in part because of their spontaneous nature and high downstream impact (e.g., GPU hangs) on the GPU.

PMU SPI Errors. Communication errors with the performance management unit (PMU) over the Serial Peripheral Interface (SPI), known as *PMU SPI errors*, can cause performance management issues (e.g., inability to change the core frequency). We observed that such errors could lead to MMU errors with a probability of 0.88 (see Figure 6), ultimately leading to user job failures. The majority of the rest of the PMU SPI errors resulted in another PMU SPI error in close succession, leading to persisting error patterns. We observed 77 occurrences of PMU SPI errors (see Table 1) with a 0.98 probability of leading to user job failures (see Table 2) on A100s.

GPU Fallen Off the Bus. GPU Fallen Off the Bus errors were logged when the GPU driver could not reach the GPU over the system bus. This error is an integration error often caused by a loose GPU-motherboard connection or contact failure because of thermal cycles [51]. Over 99% of the errors of this type lead to similar errors in close successions and eventually put the GPU into an error state.

Our *GPU hardware error propagation* analysis suggests that the error detection and recovery of GSP, PMU, and the communication interfaces (e.g., PMU SPI) need to be improved via duplications and error-detection and correction mechanisms to prevent single points of failure, as evident in A100 GPUs. In fact, AWS recommends disabling GSP for stability over performance benefits [4]. By improving GSP hardware and driver software combined with a tightly integrating CPU-GPU architecture, the H100 GPUs in the GH200 Superchips significantly improve GPU hardware resilience over the A100 GPUs. Notably, apart from three GSP errors, which follow the same propagation paths as GSP errors in A100 GPUs, we observed no other hardware error propagation paths on H100 GPUs.

4.2.3 ***NVLink Interconnect-Error Propagation.*** NVLink is an GPU-to-GPU interconnect for communication and data exchanges. An NVLink error can impact a single or multiple GPUs on the same node, possibly rendering the entire multi-GPU compute pool unavailable (see Figure 7). We observed both kinds of propagation in our error logs on A100 GPUs during the operational period on A100 GPUs.

NVLink Inter- and Intra-GPU Propagation. An NVLink error occurs when one or more NVLinks experience an error. Our analysis showed that of the 1,922 NVLink errors, 42% propagated to connected GPUs; of those errors, 17% involved three or more GPUs of the same compute node. The rest of the 1,121 NVLink errors did not propagate across GPUs. NVLink errors, like GPU hardware errors, happen in isolation without preceding errors. On the same GPU device, an NVLink error either leads to another NVLink error soon after (with a probability of 0.38) or potentially leaves the faulty GPU in an error state (with a probability of 0.61).

Although *Delta* SREs reported that most NVLink errors were benign, we found that the probability of leading to user job failure when encountered is 54% (43 cases) during the operational period on A100 GPUs. Moreover, in two incidents, GPU resets are needed to recover from critical NVLink errors, leading to over 2 hours of node interruption, and, as with GPU hardware errors, we found no

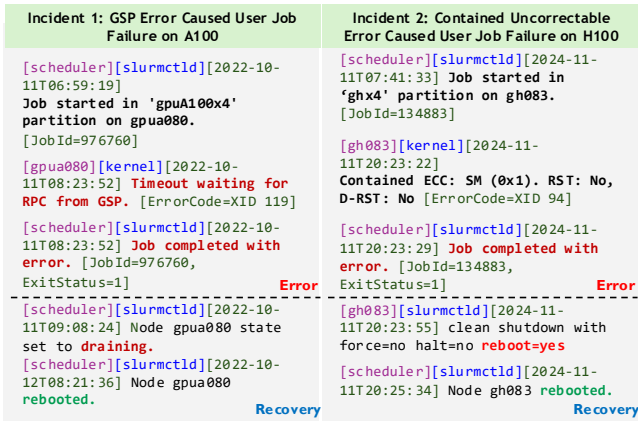


Figure 8: Incidents in which GPU errors led to job failure.

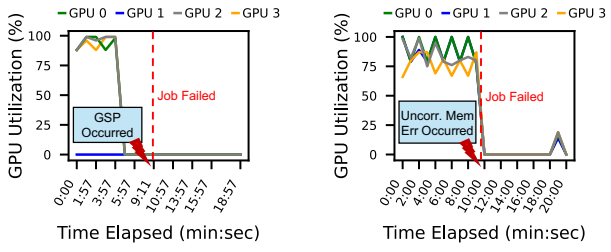
preceding hardware errors before NVLink errors, making them less predictable than memory-related errors.

We did not observe NVLink-related errors on H100 GPUs during the operational period despite observing related non-error NVLink events (e.g., link initialization). We additionally conducted NVLink tests on an *Delta*-AI node to confirm that NVLinks are enabled and fully functional. We conjecture that the improvement is due to both NVLink hardware and driver upgrades and potential changes in NVLink error logging mechanisms.

4.3 Examples of Error Propagation to User Jobs

In this section, we show how measured errors relate to NVIDIA's measured DCGM GPU utilization metric, a step towards relating GPU memory and hardware errors with their impact on the user jobs. Figure 8 presents two example incidents.

Incident 1: A GSP error stalled GPU control functions and rendered the GPU inoperable on an A100 GPU. Consequently, the user job scheduled on that GPU failed. The GSP error required the draining of the node and a full node reboot to recover, which led to the draining of all pending user jobs on that node. From the beginning of the node drain to the completion of the node reboot, the total recovery time for this incident was 23 node hours (09:08 AM to 08:21 AM the next day), during which the node was unavailable. Figure 9a shows that the corresponding GPU utilization dropped quickly following the error incident due to job failure. This incident shows that a GPU error can significantly interrupt user jobs and node availability.



(a) GSP Error **(b) Uncorr. Mem. Error**

Figure 9: GPU utilization during GPU error incidents.

Incident 2: In this incident, the H100 GPU running the user job experienced an uncorrectable ECC memory error, which led to an error containment event. The error containment event contains the uncorrectable memory error by terminating the user's job. Subsequently, a node reboot is triggered to recover from this error, resulting in a two minute node downtime. Figure 9b shows that GPU utilization dropped as the user job was terminated. This incident shows that although an uncorrectable memory error might be contained, it still results in user job termination and requires node reboot to fully recover, resulting in unexpected node downtime.

5 Propagation of Errors to Jobs

This section provides an in-depth analysis of job-level resilience and associated GPU downtime.

A GPU error can lead to:

(i) *Job Failure:* A GPU error may not be handled by jobs, either because the error itself is not contained or because there is a lack of appropriate error-handling mechanisms. For example, GSP errors lead to GPU failures and require node reboots. To recover from GPU failures, jobs need to be re-executed from the beginning or rolled back to the closest checkpoint.

(ii) *GPU and Node Downtime:* Downtime can occur when GPUs need to be reset or replaced by the operator, and no jobs can be scheduled on the GPU and the corresponding node in the interim.

5.1 Result Highlights

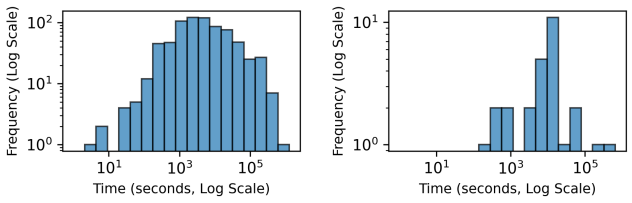
Table 2 show the overall impact of hardware errors on applications. In summary, we observe that:

(i) Except for MMU and NVLink errors, no other GPU errors are handled by jobs, thus resulting in their failure. Depending on the GPU type, the underlying cause of job failures differs. In the case of A100 GPUs, hardware errors predominantly lead to job failures, whereas memory errors are the primary cause of H100 GPUs. Therefore, there is a need to improve the resilience of underlying hardware to minimize such failures. While checkpointing is an option, checkpointing routines have a high overhead of up to 40% [31, 55, 56], including management, storage, and restore.

(ii) The overall availability per-GPU node is ~99.4% (corresponding to 9 minutes downtime/day) and ~99.3% (corresponding to 10 minutes downtime/day) for A100 and H100 GPUs respectively. This level of unavailability suggests that even if the rest of the infrastructure is highly available, current GPUs may not provide sufficient availability to meet the demands of critical applications that require greater than 3 9's of availability (downtime of 1.4 mins/day). In addition, we used emulation to project the impact of this availability distribution at increased scales (for both node scale and job duration) and found that significant overprovisioning of 5% would be necessary to handle associated failures (as explained in further detail in Section 5.4).

5.2 Job Statistics

During the characterization period, 1,420,278 user jobs were submitted to GPU nodes with a success rate of 87%. About 74% of user jobs ran on a single GPU; 24% ran on 2–4 GPUs; and only 2% of jobs used five or more GPUs.



(a) A100 (b) H100

Figure 10: Node unavailability time after GPU failures.

Because of a lack of specific information on whether jobs were ML-related, we estimated the percentage of ML user jobs based on job submission names and the system modules/libraries imported[11]. For instance, user jobs with names containing `model` or `train` were likely related to machine learning. We provide detailed statistics on node hours used, durations for both job types, and failure probabilities in Table 3. We find that across all job sizes, the failure probability lies between 6–49%, indicating a lack of adequate recovery mechanisms.

5.3 Job Failure Analysis

To understand job failure patterns, we first separated jobs into "Completed" and "GPU-Failed" depending on their completion status. Based on that job categorization, we analyzed (i) the GPU errors that were most likely lead to a job failure, and (ii) the potential recovery strategies, such as checkpointing and exception handling.

Classifying Job Runs: We classified jobs based on their exit status and proximity of job failure time to GPU error occurrence time. The job exit status was obtained from the Slurm job scheduler logs (as described in Section 3.1). We marked a job as "GPU-Failed" if a GPU error occurred within a 20-second interval before job failure.[12]

Correlating GPU Errors and Job Failures: We broke down all GPU-Failed jobs by the specific GPU errors that were most likely to lead to job failures. Table 2 provides probabilities of user job failures per GPU error for the A100 and H100 GPUs. As any of the encountered errors may have contributed to a job failure, we consider all GPU errors that occurred within the 20-second interval to be responsible for the failure.

[11]Due to privacy restrictions, we could not access the job scripts for use in job classification.

[12]Note that we do not count "zombie" jobs, i.e. jobs that have failed but not terminated by the Slurm scheduler in the analysis.

XID	GPU Error	# GPU-failed Jobs		# Jobs Encountering Given XID		Job Failure Probability (%)	
		A100	H100	A100	H100	A100	H100
31	MMU err.	3206	93	3543	126	90.48	73.80
74	NVL err.	43	0	80	0	53.75	–
122	SPI PMU RPC failure	40	0	41	0	97.56	–
119	GSP RPC timeout	31	0	31	0	100.00	–
94	Contained ECC	5	5	5	5	100.00	100.00
48	GPU DBE	0	5	0	5	–	100.00
64	Row remapping failed	0	3	0	3	–	100.00
63	Row remapping event	0	2	0	2	–	100.00

Table 2: Distribution of GPU-failed jobs across the different GPU error types for A100 and H100 GPUs. The failure probability is calculated as (# GPU-failed jobs encountering that GPU error) / (# jobs encountering that GPU error). The total number of GPU-failed jobs was 3,359 during the 895-day characterization period.

GPU Count	Count (%)	Elapsed Time (Minutes)		Failed (%)	GPU Hours (k)	
		Mean	P99		ML	Non-ML
1	1,052,993 (74.140%)	134.560	2859.677	129,395 (12.29%)	641.9	1,949.14
2-4	337,637 (23.773%)	159.839	2880.117	40,610 (12.03%)	810.4	2,749.00
5-8	13,907 (0.979%)	231.927	2880.316	4,469 (32.13%)	253.1	290.16
9-32	13,378 (0.942%)	221.636	2880.167	3,045 (22.76%)	307.5	844.40
33-64	1,375 (0.097%)	145.206	2880.017	608 (44.22%)	108.2	149.24
65-128	856 (0.060%)	320.185	2834.413	421 (49.18%)	15.1	442.70
129-256	110 (0.008%)	174.713	2041.312	7 (6.36%)	0.0	57.23
257+	22 (0.002%)	29.128	107.493	5 (22.73%)	0.0	3.44

Table 3: Job distribution, elapsed time statistics (mean, P99), failure count with percentage, and GPU hours divided into ML and non-ML categories for various A100 and H100 GPU configurations.

Overall, other than NVLink and MMU errors, all GPU errors, such as GSP RPC timeout and PMU failures, propagate (as discussed in Section 4.2) and cause job failures. Based on previous analysis, we note that hardware errors – such as GSP and PMU errors are dominant in A100 GPUs, whereas memory errors are dominant in H100 GPUs.

NVLink and MMU errors do not necessarily lead to job failures because: (1) For *NVLink errors*, the link or GPU may not be in use by any user jobs (as discussed in Section 4.2)[13]; and (2) For *MMU errors*, there can be application or library-level masking mechanisms. Besides hardware errors, MMU errors can also occur if buggy user code makes illegal memory accesses that cannot be mapped in the virtual-to-physical address space. Such errors can be managed using appropriate application-level exception handlers. Popular libraries and frameworks for machine learning [42–44] have support for handling such exceptions by skipping the associated training iteration, albeit at the cost of model quality.

5.4 Impact of GPU Downtime on Jobs

While significant node hours might be lost because of wasted compute time from failed jobs, additional node hours are also lost because of the time required to recover the impacted GPU node by either resetting it or replacing it entirely. To reset the GPU node, operators typically drain the node, i.e., wait for other jobs running on the node to complete without accepting new jobs and then reboot. After the reboot, if the node successfully passes the health check, the node reset is successful, and new jobs can be scheduled on the GPU node. If the reset is unsuccessful, the node is marked failed until the GPU is additionally tested and physically replaced, if required. To calculate the average system downtime, we estimated the total time when the GPU was unavailable, which primarily included the drain and reboot time. Figure 10 shows the distribution of the unavailable time across the entire characterization duration. Overall, we found that the expected time to service the failed node was 0.88 hours for A100 GPUs and 2.2 hours for H100 GPUs. A total of 5,700 node hours were lost to GPU downtime. Using the node downtime and failure distributions, we can estimate the availability of the GPU node as $\frac{MTTF}{MTTF+MTTR}$ equal to 99.4% and 99.3% for A100 and H100 GPUs respectively [14].

Projected impact of availability on long-running and large-scale jobs: We provide error and recovery statistics in previous sections; we

[13]Based on our understanding, NVLink and memory errors can occur even when no workload is running [16].

[14]The node MTTF number is estimated from the GPU's MTBE, for which we conservatively assume that all critical GPU errors lead to node interruption.

also attempted to project how those distributions would affect jobs running on a different system. To do so, we built a simulation tool driven by our analysis. The parameters of the simulation tool can be varied based on the scenario under consideration.

Specifically, we simulate the case in which jobs (such as ML training) use the entire set of 608 H100 GPUs and run for a duration of 1 month. These jobs require all GPUs to be operational to make progress, and frequent node failures can lead to resource unavailability and slower job progress. When such failures occur, additional provisioning of GPU resources is necessary to allow the job to resume on alternate nodes while the failed nodes recover.

The simulation uses a discrete time event simulation with node failure probabilities derived from our prior analysis. The recovery time after a failure is dependent on variables such as checkpoint load time and availability of spare GPUs. To account for the variability introduced by these factors, we parameterize recovery time and perform a parameter sweep. For a training job with 608 GPUs and recovery time of 2.2 hours, the required overprovisioning is 5%: i.e., 31 additional GPUs are needed beyond the original 608 to maintain availability of 99.9% at the job level. While at first glance, such overprovisioning would appear to be a small cost, for the above example, it would cost over $1 million per month for a 1000 node cluster (our analysis is based on AWS H100 GPU rental rates [10]). However, if the recovery time is reduced to 5 minutes, downtime decreases significantly, and the required overprovisioning drops to 2%, a 2.5× reduction. This highlights the criticality of minimizing recovery time to reduce downtime for large and long-running jobs.

In summary, every GPU node in the system has "two nines" of availability. While such availability does not significantly impact small jobs that use recovery mechanisms, large jobs can face significant downtime. Significant overprovisioning of up to 5% is required to eliminate such downtime.

6 Discussion

Justification of Analyzing Errors. Data-driven HPC resilience characterization studies analyze operational data on system/application errors to provide insights into system resilience [11, 12, 18, 19, 32, 33, 38, 39, 51, 52]. While in those studies, the error rate is used as the key metric to quantify resilience, some [5, 50] argue that fault rate is a more appropriate metric. Errors represent the manifestation of faults and have direct downstream consequences, such as triggering of recovery mechanisms, application interruptions, or system-wide outages (SWOs). While a fault may result in multiple errors, it is the resulting errors that the recovery mechanisms must address to maintain system health. These errors and their recovery process directly impact system health, performance, and availability. Thus, SREs prioritize errors over faults. Hence, like many others who study operational data, we chose to study errors.

Reliability of Logging. A potential source of error is logging inconsistency from (i) missing logs due to storage failures, (ii) node lockups where the NVIDIA driver fails to log the error prior to the failure, and (iii) incorrect job-status captured by Slurm. However, these issues negligibly affected our analysis because: (1) *Delta* minimizes storage failure impact by streaming logs in real time to centralized storage; (2) SRE records show only 27 A100 and nine H100 node lockups–0.18% and 0.49% of total GPU errors–making their impact negligible; and (3) while eliminating false-negative job status is challenging, GPU errors studied led to job failure almost 100% of times (Table 2), indicating failure statuses were reliably captured by Slurm and false-negatives minimally impacted our findings.

7 Related Work

Existing work has analyzed GPU resilience at the microarchitecture, cluster, and application levels. This paper extends existing work via comprehensive analyses of GPU error characteristics, propagation, and impact on user jobs.

Microarchitecture-level GPU Resilience. Previous research [21, 46, 54, 59] has primarily focused on the resilience of individual GPUs at the microarchitecture and software levels, e.g., for older generations such as the NVIDIA G80 [7]. However, the earlier work did not evaluate the resiliency of modern GPUs in large-scale HPC settings.

System-level GPU Resilience in HPC Settings. Existing studies have analyzed the resilience of GPUs in HPC systems [8, 11, 17, 20, 25], for example, the NVIDIA Tesla K20X GPUs in various supercomputers [12, 18, 19, 32, 33, 39, 51, 52]. Studies of the Blue Waters, Titan, and Summit supercomputers [12, 25, 32, 33, 38] have examined node failures and GPU error characteristics. However, such work either studied previous generations of GPUs with a focus on GPU memory or on cluster-level resilience instead of GPU resilience. Our work complements those by providing resilience insights into the latest generation of GPUs, focusing on a broader range of components.

Application-level GPU Resilience in Data-centers and Deep Learning Workloads. Recent research has focused on understanding GPU power usage [33], GPU component-level failures [24], software-level error handling [15, 29, 30], and the impact of GPU software error propagation on GPGPU applications [3, 27, 45, 57] and emerging GPU workloads such as convolutional neural networks (CNNs) [9, 13], large language models (LLMs) [14, 23], and privacy/safety-critical applications [28, 40, 47].

8 Conclusion

This paper describes the results of a resilience study of *Delta*, which consists of 1,056 NVIDIA A100 and H100 GPUs. The study used up to 2.5 years of operational data on GPU errors collected across those GPUs. We assessed the resilience of GPU components, error propagation paths, and impact on jobs and compared the two generations of GPUs. In future work, we will extend the analysis presented to other accelerators and larger-scale systems, running more complex HPC and ML workloads.

Acknowledgments

We thank the reviewers for valuable feedback and J. Applequist, H. C. Fairow, S. Weick, and K. Atchley for support. This work used NCSA *Delta* and *DeltaAI* compute resources supported by NSF grants 2005572 and 2320345, with allocations from ACCESS (NSF 2138259, 2138286, 2138307, 2137603, 2138296), and additional support from NSF grants 2029049, 2319190, 2430244, 2530738, the IBM-Illinois Discovery Accelerator Institute, and the VinUni-Illinois Smart Health Center. Any opinions expressed here are those of the authors and do not reflect the views of NSF, IBM, or VinUni.

References

[1] Md Mahbub Alam, Luis Torgo, and Albert Bifet. 2022. A survey on spatio-temporal data analytics systems. *Comput. Surveys* 54, 10s (2022), 1–38.

[2] Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[3] Abdul Rehman Anwer, Guanpeng Li, Karthik Pattabiraman, Michael Sullivan, Timothy Tsai, and Siva Kumar Sastry Hari. 2020. GPU-Trident: Efficient modeling of error propagation in GPU programs. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[4] AWS Official. 2023. *How do I troubleshoot Xid errors on my NVIDIA GPU-accelerated EC2 Linux instance?* https://repost.aws/knowledge-center/ec2-linux-troubleshoot-xid-errors

[5] Majed Valad Beigi, Yi Cao, Sudhanva Gurumurthi, Charles Recchia, Andrew Walton, and Vilas Sridharan. 2023. A Systematic Study of DDR4 DRAM Faults in the Field. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 991–1002. doi:10.1109/HPCA56546.2023.10071066

[6] Brett Bode, Gregory Bauer, Laura Herriott, Volodymyr Kindratenko, and William Gropp. 2025. DeltaAI: A National Resource for AI/ML Research. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration (PEARC '25)*. Association for Computing Machinery, New York, NY, USA, Article 53, 4 pages. doi:10.1145/3708035.3736062

[7] Giani Braga, Marcio M Gonçalves, and José Rodrigo Azambuja. 2023. Software-controlled pipeline parity in GPU architectures for error detection. *Microelectronics Reliability* 148 (2023), 115155.

[8] Nevin Cini and Gulay Yalcin. 2020. A Methodology for Comparing the Reliability of GPU-Based and CPU-Based HPCs. *ACM Comput. Surv.* 53, 1, Article 22 (Feb. 2020), 33 pages. doi:10.1145/3372790

[9] Josie E Rodriguez Condia, Juan-David Guerrero-Balaguera, Fernando F Dos Santos, Matteo Sonza Reorda, and Paolo Rech. 2022. A multi-level approach to evaluate the impact of GPU permanent faults on CNN's reliability. In *2022 IEEE International Test Conference (ITC)*. IEEE, 278–287.

[10] DataCrunch. 2024. Cloud GPU Pricing Comparison in 2025. https://datacrunch.io/blog/cloud-gpu-pricing-comparison Accessed: 2025-04-14.

[11] Nathan DeBardeleben, Sean Blanchard, Laura Monroe, Phil Romero, Daryl Grunau, Craig Idler, and Cornell Wright. 2014. GPU behavior on a large HPC cluster. In *Euro-Par 2013: Parallel Processing Workshops: BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers 19*. Springer, 680–689.

[12] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. 2014. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 610–621. doi:10.1109/DSN.2014.62

[13] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. 2018. Analyzing and increasing the reliability of convolutional neural networks on GPUs. *IEEE Transactions on Reliability* 68, 2 (2018), 663–677.

[14] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[15] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2014. GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 221–230.

[16] NVIDIA Forums. 2024. NVLink error 74 fatal error detected. https://forums.developer.nvidia.com/t/nvlink-error-74-fatal-error-detected/55582/1 Accessed: 2025-02-24.

[17] Haryadi S Gunawi, Riza O Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, et al. 2018. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage (TOS)* 14, 3 (2018), 1–26.

[18] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[19] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. 2015. Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 37–44.

[20] Imran S Haque and Vijay S Pande. 2010. Hard data on soft errors: A large-scale assessment of real-world error rates in GPGPU. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 691–696.

[21] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W Keckler, and Joel Emer. 2015. SASSIFI: Evaluating resilience of GPU applications. In *Proceedings of the Workshop on Silicon Errors in Logic-System Effects (SELSE)*.

[22] Chip Huyen. 2022. *Designing Machine Learning Systems*. O'Reilly Media, Inc.

[23] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.

[24] Jeageun Jung and Mattan Erez. 2023. Predicting Future-System Reliability with a Component-Level DRAM Fault Model. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 944–956.

[25] Apostolos Kokolis, Michael Kuchnik, John Hoffman, Adithya Kumar, Parth Malani, Faye Ma, Zachary DeVito, Shubho Sengupta, Kalyan Saladi, and Carole-Jean Wu. 2025. Revisiting Reliability in Large-Scale Machine Learning Research Clusters . In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 1259–1274. doi:10.1109/HPCA61900.2025.00096

[26] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024. LLM inference serving: Survey of recent advances and opportunities. *arXiv preprint arXiv:2407.12391* (2024).

[27] Guanpeng Li, Karthik Pattabiraman, Chen-Yang Cher, and Pradip Bose. 2016. Understanding Error Propagation in GPGPU Applications. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 240–251. doi:10.1109/SC.2016.20

[28] Zilinghan Li, Shilan He, Ze Yang, Minseok Ryu, Kibaek Kim, and Ravi Madduri. 2025. Advances in Appfl: a Comprehensive and Extensible Federated Learning Framework . In *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–11. doi:10.1109/CCGRID64434.2025.00031

[29] Xinyu Lian, Sam Ade Jacobs, Lev Kurilenko, Masahiro Tanaka, Stas Bekman, Olatunji Ruwase, and Minjia Zhang. 2025. Universal Checkpointing: A Flexible and Efficient Distributed Checkpointing System for Large-Scale DNN Training with Reconfigurable Parallelism. In *Proceedings of the 2025 USENIX Annual Technical Conference*.

[30] Naoya Maruyama, Akira Nukada, and Satoshi Matsuoka. 2010. A high-performance fault-tolerant software framework for memory on commodity GPUs. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 1–12.

[31] Avinash Maurya, Robert Underwood, M Mustafa Rafique, Franck Cappello, and Bogdan Nicolae. 2024. DataStates-LLM: Lazy asynchronous checkpointing for large language models. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*. 227–239.

[32] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. 2016. A large-scale study of soft-errors on GPUs in the field. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 519–530.

[33] Bin Nie, Ji Xue, Saurabh Gupta, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2017. Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 22–31.

[34] NVIDIA. 2023. *NVIDIA Data Center GPU Driver version 525.105.17 (Linux) / 528.89 (Windows)*. https://docs.nvidia.com/datacenter/tesla/pdf/NVIDIA_Data_Center_GPU_Driver_Release_Notes_525_v3.0.pdf

[35] NVIDIA. 2024. *GPU Deployment and Management*. https://docs.nvidia.com/deploy/pdf/XID_Errors.pdf

[36] NVIDIA Corporation. 2023. *NVIDIA Grace Hopper Superchip Architecture Whitepaper*. https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

[37] NVIDIA Corporation. 2024. *NVIDIA GPU Memory Error Management*. https://docs.nvidia.com/deploy/a100-gpu-mem-error-mgmt/index.html Release r555.

[38] Vladyslav Oles, Anna Schmedding, George Ostrouchov, Woong Shin, Evgenia Smirni, and Christian Engelmann. 2024. Understanding GPU Memory Corruption at Extreme Scale: The Summit Case Study. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 188–200.

[39] George Ostrouchov, Don Maxwell, Rizwan A Ashraf, Christian Engelmann, Mallikarjun Shankar, and James H Rogers. 2020. GPU lifetimes on Titan supercomputer: Survival analysis and reliability. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–14.

[40] Jon Perez-Cerrolaza, Jaume Abella, Leonidas Kosmidis, Alejandro J Calderon, Francisco Cazorla, and Jose Luis Flores. 2022. GPU devices for safety-critical systems: A survey. *Comput. Surveys* 55, 7 (2022), 1–37.

[41] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. 2005. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry* 26, 16 (2005), 1781–1802.

[42] PyTorch Contributors. 2017. *PyTorch Issue #1137: How to Handle Corrupted Data in Dataloader*. https://github.com/pytorch/pytorch/issues/1137 Accessed: 2024-11-26.

[43] PyTorch Contributors. 2018. *Discussion: How to Handle Exception in DistributedDataParallel.* https://discuss.pytorch.org/t/how-to-handle-exception-in-distributeddataparallel/42026 Accessed: 2024-11-26.

[44] PyTorch Lightning Contributors. 2021. *PyTorch Lightning Discussion: Handling Exceptions in Forward Pass.* https://github.com/Lightning-AI/pytorch-lightning/discussions/15188 Accessed: 2024-11-26.

[45] Paolo Rech, Laércio Lima Pilla, Philippe Olivier Alexandre Navaux, and Luigi Carro. 2014. Impact of GPUs parallelism management on safety-critical and HPC applications reliability. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 455–466.

[46] Dimitris Sartzetakis, George Papadimitriou, and Dimitris Gizopoulos. 2022. gpuFI-4: A microarchitecture-level framework for assessing the cross-layer resilience of Nvidia GPUs. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 35–45.

[47] Deval Shah, Zi Yu Xue, Karthik Pattabiraman, and Tor M Aamodt. 2024. Characterizing and Improving Resilience of Accelerators to Memory Errors in Autonomous Robots. *ACM Transactions on Cyber-Physical Systems* 8, 3 (2024), 1–33.

[48] Gilad Shainer, Tong Liu, John Michalakes, Jacob Liberman, Jeff Layton, Onur Celebioglu, Scot A Schultz, Joshua Mora, and David Cownie. 2009. Weather research and forecast (WRF) model performance and profiling analysis on advanced multi-core HPC clusters. In *10th LCI International Conference on High-Performance Clustered Computing*.

[49] David Skinner and William Kramer. 2005. Understanding the causes of performance variability in HPC workloads. In *Proceedings of the IEEE International Workshop/Symposium on Workload Characterization*. IEEE, 137–149.

[50] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. 2015. Memory Errors in Modern Systems: The Good, The Bad, and The Ugly. *SIGPLAN Not.* 50, 4 (March 2015), 297–310. doi:10.1145/2775054.2694348

[51] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge Leadership Computing Facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[52] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, et al. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 331–342.

[53] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[54] Alessandro Vallero, Sotiris Tselonis, Dimitris Gizopoulos, and Stefano Di Carlo. 2018. Multi-faceted microarchitecture level reliability characterization for NVIDIA and AMD GPUs. In *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 1–6.

[55] Borui Wan, Mingji Han, Yiyao Sheng, Yanghua Peng, Haibin Lin, Mofan Zhang, Zhichao Lai, Menghan Yu, Junda Zhang, Zuquan Song, et al. 2024. ByteCheckpoint: A Unified Checkpointing System for Large Foundation Model Development. *arXiv preprint arXiv:2407.20143* (2024).

[56] Long Wang, Karthik Pattabiraman, Zbigniew Kalbarczyk, Ravishankar K Iyer, Lawrence Votta, Christopher Vick, and Alan Wood. 2005. Modeling coordinated checkpointing for large-scale supercomputers. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 812–821.

[57] Xiaohui Wei, Hengshan Yue, Shang Gao, Lina Li, Ruyu Zhang, and Jingweijia Tan. 2020. G-SEAP: Analyzing and characterizing soft-error aware approximation in GPGPUs. *Future Generation Computer Systems* 109 (2020), 262–274.

[58] Ronglong Wu, Shuyue Zhou, Jiahao Lu, Zhirong Shen, Zikang Xu, Jiwu Shu, Kunlin Yang, Feilong Lin, and Yiming Zhang. 2024. Removing Obstacles before Breaking Through the Memory Wall: A Close Look at HBM Errors in the Field. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 851–867. https://www.usenix.org/conference/atc24/presentation/wu-ronglong

[59] Lishan Yang, George Papadimitriou, Dimitris Sartzetakis, Adwait Jog, Evgenia Smirni, and Dimitris Gizopoulos. 2024. GPU Reliability Assessment: Insights Across the Abstraction Layers. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–13.

[60] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, 44–60.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

All artifacts and instructions can be found in the following link on Zenodo: **https://doi.org/10.5281/zenodo.15287639.** To simplify the evaluation of artifacts, we provide pre-processed intermediate output as pickle files and instructions to reproduce the results for the listed artifacts in the README. **Please see the README in the Zenodo repository for detailed and efficient evaluation instructions.**

A Overview of Contributions and Artifacts

A.1 Paper's Main Contributions

By studying the GPU resilience of *Delta*[15], our work contributes to the following areas: (i) compares GPU memory and hardware resilience via error statistics analysis of H100 and A100 GPUs; (ii) characterizes intra- and inter-GPU error propagation graphs for GPU memory, hardware, and NVLink error categories, and presents real-world user job failure examples; and (iii) analyzes GPU error impacts on user job success rate and node availability, and project overprovisioning costs for large-scale, critical workloads, via emulation.

For the GPU memory and hardware resilience comparisons, our contributions are as follows:

- **C_1** H100 GPU memory resilience is worse than that of A100 GPU memory in terms of both per-GPU mean time between errors (MTBE) and per-node MTBE. Specifically, H100 shows 3.2× lower per-GPU MTBE and 24% lower per-GB MTBE compared to A100 for uncorrectable ECC memory errors.
- **C_2** Although H100's memory error-recovery mechanisms mitigated (e.g., by memory row remapping) 92% of uncorrectable ECC memory errors, they were insufficient to handle the increase in memory capacity and the corresponding increase in row remapping events on the H100 GPUs, as evident by row-remapping failures.
- **C_3** H100 GPUs demonstrated significantly improved hardware resilience over A100 GPUs with respect to critical components such as GSP, NVLink, and PMU SPI, which were major sources of job failures on A100 GPU nodes.

For A100 and H100 GPU error propagation path characterization, our contributions are as follows:

- **C_4** We generated detailed error propagation graphs for critical GPU errors in (i) GPU memory, (ii) GPU hardware, and (iii) the NVLink category (Figures 5-7). The error propagation analysis showed that while the H100 GPUs' error propagation paths were GPU memory error-related, A100 GPUs' error propagation paths were predominantly GPU hardware error-related.
- **C_5** Our analysis showed that GPU hardware error-related propagations on A100 GPUs were a single point of failure as they appeared in isolation from other errors and were harder to predict.
- **C_6** We present three incidents of GPU error-caused user job failures in Figures 1 and 8. For the two incidents presented in Figure 8, we additionally included the corresponding GPU utilization during those job failures.

We also make the following contributions to characterizing the impact of GPU errors on user job success rate, node availability, and overprovisioning costs:

- **C_7** GPU errors on both A100 and H100 GPUs frequently resulted in job failures due to the lack of robust recovery mechanisms at the application level. Except for MMU and NVLink errors, GPU errors cannot be handled by application-level mechanisms, resulting in a close to 100% job failure rate. The underlying causes of job failures differ by GPU type: hardware errors were the predominant cause in A100 GPUs, whereas memory errors were the primary cause in H100 GPUs.
- **C_8** The overall availability per GPU node was approximately 99.4% for A100 GPUs and 99.3% for H100 GPUs, corresponding to a downtime of 9–10 minutes per day.
- **C_9** To maintain 99.9% availability at the job level, overprovisioning of 5% would be necessary, introducing significant cost overheads of $1 million per month. If GPU node availability were improved to 99.9%, the required overprovisioning would be reduced by 2.5×, significantly reducing the overprovisioning costs.

A.2 Computational Artifacts

The computational artifacts of this work can be accessed via [link]. These artifacts constitute a data pipeline that consumes system and user job-related monitoring data and outputs the statistical, propagation, and application-impact analysis outlined in Section 3 of the paper.

The input to our data pipeline is outlined in Section 2 of the paper; it consists of Linux system logs, Slurm scheduling entries, DCGM time series, and user job submission information. Since those input data contain sensitive system and user information, such as user identifiers, email addresses, project names, private IPs, and system identifiers, we do not have permission from the system admins to make them public. We are actively working with the system admins on sanitizing and anonymizing the input data; we plan to provide the input data and detailed instructions for executing our data pipeline for artifact evaluation upon paper acceptance. Note that our data pipeline is generalizable to other systems with minimal adaptation effort, since NVIDIA XID errors are standard GPU errors and are likely to follow the same format across different systems. The following is a list of computational artifacts used in this work with DOI: *https://doi.org/10.5281/zenodo.15287639* on Zenodo:

- **A_1** `SystemLogAnalysisPipeline.py` [link]
- **A_2** `JobDetailCollect.sh` [link]
  `ApplicationStatsPipeline.ipynb` [link]
- **A_3** `DCGMUtilizationAnalysis.py` [link]
- **A_4** `OverprovisioningEmulation.py` [link]

The above artifacts were executed on input data obtained from *Delta* covering 2.5 years of operation, including 11.7M total GPU

[15] *Delta* is an HPC system operated by the National Center for Supercomputing Applications (NCSA) at the University of Illinois Urbana-Champaign.

hours across the 1,056 A100 and H100 GPUs. Because of the size of our data, the end-to-end execution time of the artifacts is approximately 675 minutes on a 28-core machine. Note that every one of the above artifacts produced multiple data points, which helped us produce multiple table entries, figures, and graphs, which collectively constitute our findings, as shown in the following table. We will now explain in detail each artifact's relationship to our contribution.

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1, C_2, C_3, C_4, C_5	Table 1 Figures 5-7
A_1, A_2, A_3	C_6	Figures 1, 8, and 9
A_1, A_2	C_7, C_8	Tables 2 and 3 Figure 10
$A_1, A_2, A_4,$	C_9	

B Artifact Identification

B.1 Computational Artifact A_1

Relation To Contributions

Artifact A_1 (`SystemLogAnalysisPipeline.py`) implements the data processing pipeline for *Delta* system logs over the 2.5-year measurement period and provides the following functionalities: (i) system log preprocessing, including XID error regex matching, error keyword matching, and error coalescing; (ii) GPU error aggregation and statistics analysis; and (iii) error propagation path analysis. These functionalities together provide foundations for MTBE calculations and error propagation path constructions, which we used to generate contributions C_1, C_2, C_3, C_4, C_5, and partly for C_6 and C_9.

Expected Results

The expected results of this artifact are (i) Pickle[16] files containing the regex-matched XID errors that occurred each day during the measurement period, before primary error keyword matching and coalescing (without de-duplication); (ii) Pickle files after the keyword matching and error coalescing processes (de-duplication), per day; (iii) an aggregated Pickle file containing error count statistics for each XID event during the entire measurement period, obtained by aggregating per-day pickle files; (iv) Pickle files for intra-GPU error propagation analysis in the form of a list of source and resulting errors pairs, with propagation time recorded; and (v) Pickle files for inter-GPU error propagation for NVLink errors. All these items were generated for both H100 GPUs and A100 GPUs across their corresponding measurement periods.

Expected Reproduction Time (in Minutes)

The expected computation time for A_1 on the 2.5 years of measurement data is 600 minutes on a 28-core machine consisting of 2 Intel E5-2660v4 CPUs and 128 GB of system RAM.

[16] Pickle files are generated by the Python Pickle Package.

Artifact Setup (incl. Inputs)

Hardware. The hardware requirements are identical across all artifacts and are as follows: memory, 128 GB; CPU, 2×14-core Intel Xeon E5-2660 v4 CPU.

Software. The software environment is identical across all artifacts and is as follows:

- System: Red Hat Enterprise Linux 9.5 (Plow)
- Python Interpreter: `Python 3.12`
- Python Packages used:
  - `tqdm`
  - `google-re2`
  - `pandas`
  - `matplotlib`
  - `hyperscan`
  - `jupyter`
  - `numpy`
- We also used the following Python standard packages:
  - `subprocess`
  - `multiprocessing`
  - `itertools`
  - `pickle`
  - `statistics`
  - `sys`
  - `os`
  - `datetime`
  - `math`
  - `collections`

Datasets / Inputs. The inputs to A_1 are the per-day system logs compressed in `*.gz` format. Artifact A_1 will automatically decompress and process all available log files in a user-specified time range and data path.

Installation and Deployment. Upon installation of all the required Python packages, the script can be run using the following example commands:

```
python3 SystemLogAnalysisPipeline pkl 2022 2026 1 13 1 32 \
&& python3 SystemLogAnalysisPipeline match_keyword "" \
&& python3 SystemLogAnalysisPipeline dedupe "" 5 \
&& python3 SystemLogAnalysisPipeline agg_specific -1 gpu "deduped_5s
      "

python3 split_to_diff_gpus_artifact.py \
&& python3 SystemLogAnalysisPipeline
      cond_prob_forward_diff_pci_timediff 2022 2026 1 13 1 32
      same_dev_id_same_node True _deduped_5s 5 \
&& python3 SystemLogAnalysisPipeline
      cond_prob_forward_diff_pci_timediff 2022 2026 1 13 1 32
      same_node_diff_dev_id True _deduped_5s 5 \
&& python3 SystemLogAnalysisPipeline agg_cond_prob_pci_timediff -1
      forward same_dev_id_same_node True _deduped_5s 5 \
&& python3 SystemLogAnalysisPipeline agg_cond_prob_pci_timediff -1
      forward same_node_diff_dev_id True _deduped_5s 5
```

This set of commands performs data preprocessing, generates error statistics for the H100 and A100 GPUs separately, and then generates the propagation analysis. Detailed instructions and supported arguments will be released with the Artifact Evaluation Appendix.

Artifact Execution

The artifact consists of four major tasks: T_1, T_2, T_3, and T_4. T_1 decompresses the gz compressed raw system log files per day, performs regex matching for extracting XID-related error logs, and generates Expected Result item (i) for each day. T_2 uses the output from T_1 and further performs XID error keyword matching and error coalescing to remove duplicated error logs that originated in the same error; the error-coalescing window Δt is set to 5 seconds. A detailed justification of $\Delta t = 5$ is in the paper's methodology section (Section 3.2). The result of T_2 is item (ii) in the Expected Results section. T_3 aggregates the per-day processed errors into a single error list, separated by GPU types (A100 vs. H100). Then T_4 uses the outcome of T_3 to generate intra- and inter-GPU propagation analysis by implementing the error propagation analysis algorithm discussed in methodology section (Section 3.2). Note that the error propagation window cut-off Δt is set to 5 seconds; a detailed justification is presented in the paper.

Artifact Analysis (incl. Outputs)

The count statistics generated from item (iii) of the Expected Results for the GPU error types are expected to match the counts in Table 1, and the corresponding system-wide and per-node MTBEs are then computed by the method specified in the paper's methodology section (Section 3.2). Moreover, items (iv) and (v) generate the propagation figures (Figures 5-7) for GPU memory, GPU hardware, and NVLink errors. The error propagation probability and propagation time are extracted by aggregating all errors of the same XID and should match those presented in the error propagation figures. Because of noise in error propagation analysis, we omit the edges for which the propagation probability is less than 1%.

B.2 Computational Artifact A_2

Relation To Contributions

Artifact A_2 consists of the script (`JobDetailCollect.sh`), which uses `sacct` to collect job metadata, including user information, resource usage, runtime behavior, job completion status, and the pipeline (`ApplicationStatsPipeline.ipynb`), which is used for analyzing the impact of GPU errors on job resilience during the 2.5-year characterization period. A_2 provides the following functionalities: (i) job data collection using `sacct`; (ii) job failure classification based on correlation with GPU error events; (iii) job distribution across GPU counts, runtime statistics, GPU-hour breakdown by ML and non-ML, and failure analysis; and (iv) GPU unavailability estimation from drain and recovery durations. These functionalities enable a comprehensive understanding of how GPU reliability impacts job resilience, resource usage, and system availability and are used for contributions C_6, C_7, C_8, and partly C_9.

Expected Results

The expected results of this artifact are (i) files containing job metadata collected using `sacct` that capture job resource usage, runtime characteristics, and execution state; (ii) files containing classified job records as GPU-failed or completed based on temporal correlation with GPU error logs; (iii) a table summarizing job statistics including GPU count, runtime (mean and P99), failure rates, and estimated GPU-hours for ML and non-ML workloads; and (iv) separate visualizations for A100 and H100 GPUs showing the distribution of node draining and recovery times.

Expected Reproduction Time (in Minutes)

The expected reproduction time of this artifact is approximately 75 minutes on the hardware stated for A_1.

Artifact Setup (incl. Inputs)

Hardware. Same as listed for A_1.

Software. Same as listed for A_1.

Datasets / Inputs. The input to this artifact consists of (i) job metadata files collected using `sacct`, stored in TXT (text file) format; and (ii) GPU error logs recorded over the characterization period. All files are preprocessed and analyzed within the ipynb notebook based on a user-specified data directory and date range.

Installation and Deployment. Upon installation of all required Python packages, the Jupyter Notebook can be executed in any Python 3.8+ environment.

Artifact Execution

The artifact consists of four major tasks: T_1, T_2, T_3, and T_4. T_1 collects job-level metadata using `sacct` from the *Delta* Slurm scheduler database and stores job records, including runtime, resource allocation, and execution status, to text files. T_2 uses the output from T_1 along with error classifications from A_1 to identify GPU-induced job failures based on temporal correlation with GPU error events, as outlined in the paper's methodology section (Section 3.2). This analysis generates the job failure statistics reported in Table 2. T_3 processes the output of T_1 to extract aggregate job statistics, such as GPU count, runtime characteristics (mean and P99), job failure rates, and GPU hours by ML and non-ML jobs. These results correspond to Table 3 in the paper. T_4 combines the output from T_1 and T_2 to estimate node recovery time after GPU failures, generating the data illustrated in Figure 10. In addition, we derive the per-node availability using both the GPU error and failure statistics generated by A_1 and the node recovery time after GPU failures estimated by A_2.

Artifact Analysis (incl. Outputs)

The classified job failure records generated by T_2 are expected to match the GPU error-induced failure statistics shown in Table 2. The aggregate job metrics computed in T_3 should match the job usage breakdowns and failure distributions in Table 3. The node unavailability times derived in T_4 should reproduce the distributions shown in Figure 10 for both A100 and H100 GPUs.

B.3 Computational Artifact A_3

Relation To Contributions

A_3 contains a Python script (`DCGMUtilizationAnalysis.py`) that reads the utilization data between the start date and the end date from the *Delta* DCGM database and visualizes the utilization of

four GPUs in the same node before and after a job failure occurs. These visualizations are used to construct the contribution C_6.

Expected Results

The expected result of this artifact is a figure visualizing the utilization changes in four GPUs within a specific node 10 minutes before and after a job running on either of those GPUs encounters a failure. The x-axis is the time elapsed starting from 10 minutes before the failure occurs until 10 minutes after the failure, and the y-axis is the GPU utilization from 0% to 100%. This figure is expected to be the same as Figure 9.

Expected Reproduction Time (in Minutes)

The expected computation time for the artifact is less than a minute.

Artifact Setup (incl. Inputs)

Hardware. Same as listed for A_1.

Software. Python 3.12.7, and compatible matplotlib, numpy, pandas.

Datasets / Inputs. The input to this artifact is the per-day DCGM data collected and stored in .csv format.

Artifact Execution

The artifact consists of a single task T_1. T_1 reads the utilization data from the DCGM data and visualizes the utilization of four GPUs in a targeted node. The output of T_1 is a figure similar to Figure 9.

Artifact Analysis (incl. Outputs)

The output data generated from this artifact should match Figure 9 and contribution C_6.

B.4 Computational Artifact A_4

Relation To Contributions

A_4 contains a Python script (`OverprovisioningEmulation.py`) to emulate job availability based on GPU availability, cluster size, and overprovisioning. GPU failures are emulated in discrete time steps with a random number generator. These failures are translated into job availability values after accounting for the ability to handle failures based on overprovisioning. Additionally, a binary search estimates the minimal overprovisioning required to satisfy the job availability target.

Expected Results

Minimum required overprovisioning values are outputted based on the input availability, job size, failure, and recovery distributions.

Expected Reproduction Time (in Minutes)

Artifact Setup (incl. Inputs)

Hardware. Same as listed for A_1.

Software. Python 3.12.7

Datasets / Inputs. Failure and recovery distributions from A_1

Installation and Deployment. The artifact can be executed as a Python script with constants defined in the header.

Artifact Execution

The artifact has two major tasks: T_1 and T_2. T_1 emulates GPU downtime via a random sample process given the GPU error statistics generated from A_1 and generates a time distribution of GPU node unavailability. T_2 then uses this distribution and the specified job availability target to compute the overprovisioning requirement, contributing to C_9

Artifact Analysis (incl. Outputs)

The output generated from the artifact will match the overprovisioning numbers mentioned in Section 5 of the paper.

Artifact Evaluation (AE)

All artifacts and instructions can be found in the following link on Zenodo: **https://doi.org/10.5281/zenodo.15287639.** To simplify the evaluation of artifacts, we provide pre-processed intermediate output as pickle files and instructions to reproduce the results for the listed artifacts in the README. The following sections provide additional guidance for adapting the analysis pipeline to new systems or for starting from raw system logs. **Please see the README in the Zenodo repository for detailed and efficient evaluation instructions.**

C.1 Computational Artifact A_1

Artifact Setup (incl. Inputs)

Artifact A_1 (*SystemLogAnalysisPipeline.py*) processes raw system log files (hereafter referred to as "raw system logs") and generates a list of artifacts (see Section B.1) that are essential for downstream GPU resilience analysis.

To set up the runtime environment for Artifact A_1 to **analysis raw system logs**, follow these steps:

(1) Provision a machine with hardware specifications comparable to or exceeding those listed in AD Section B.1 for processing logs at large-scale.
(2) Install Python version 3.12 or later on a system running either Red Hat Enterprise Linux 9.5 or Ubuntu 20.04.
(3) Install the required Python packages using `pip3` or `conda` (e.g., `pip3 install <packagename>==<version>`).
   - Python packages required:
     - `tqdm`
     - `google-re2`
     - `pandas`
     - `matplotlib`
     - `hyperscan`
     - `jupyter`
     - `numpy`

The inputs to Artifact A_1 are as follows:

(1) A regular expression file, `regex_with_new_gpu.tsv` (provided), used to match and parse XID-related log lines, and to filter out non-XID entries.
(2) A directory containing the raw system log files. Each log file path must follow a directory structure of the form `<basedir>/yyyy/yyyymmdd`. For example, the log file for March 15, 2023, should be located at `<basedir>/2023/20230315`.

Each log line consists of the following space separated fields:

```
timestamps hostname process_info log_message
```

An example log line is shown below:

```
2024-08-23T01:21:09.212665-05:00 host1 kernel: NVRM: Xid (PCI
     :0000:01:00): 48, pid='<unknown>',name=<unknown>, An
     uncorrectable double bit error (DBE) has been detected on GPU
     in the framebuffer at physAddr 0x000000000 partition 0,
     subpartition 0.
```

Since Artifact A_1 is designed to process system log files from any HPC environment that conforms to the specified directory structure and log format, we encourage the broader community to adapt and apply A_1 to their own systems.

Artifact Execution

To execute Artifact A_1, set `REGEX_FILE_PATH` to the full path of the `regex_with_new_gpu.tsv` file, and `ATLAS_LOG_FOLDER_PATH` to the directory (`basedir`) containing the raw system logs. Then, run the following commands:

```
python3 SystemLogAnalysisPipeline pkl 2022 2026 1 13 1 32

python3 SystemLogAnalysisPipeline match_keyword ""

python3 SystemLogAnalysisPipeline dedupe "" 5

python3 SystemLogAnalysisPipeline agg_specific -1 gpu "deduped_5s"

python3 split_to_diff_gpus_artifact.py

python3 SystemLogAnalysisPipeline \
     cond_prob_forward_diff_pci_timediff 2022 2026 1 13 1 32 \
     same_dev_id_same_node True _deduped_5s 5

python3 SystemLogAnalysisPipeline \
     cond_prob_forward_diff_pci_timediff 2022 2026 1 13 1 32 \
     same_node_diff_dev_id True _deduped_5s 5

python3 SystemLogAnalysisPipeline agg_cond_prob_pci_timediff -1 \
     forward same_dev_id_same_node True _deduped_5s 5

python3 SystemLogAnalysisPipeline agg_cond_prob_pci_timediff -1 \
     forward same_node_diff_dev_id True _deduped_5s 5
```

The above commands correspond to the following analysis procedure:

(1) Extract XID error logs from raw system logs by filtering and matching each log line using the provided XID regular expressions. This step generates intermediate pickle files. The six numerical inputs represent `start_year`, `end_year`, `start_month`, `end_month`, `start_day`, and `end_day`, respectively. For example, the provided command processes logs from January 1, 2022, to December 31, 2025.
(2) Identify the primary XID error log for each XID error event using keyword matching. Some XID errors generate a single primary log entry accompanied by multiple secondary entries, which provide additional context such as memory addresses or channel identifiers.
(3) De-duplicate the extracted XID error logs through error coalescing, using a 5-second coalescing window.
(4) Aggregate all daily pickle files into a single, unified pickle file for downstream analysis.
(5) Partition the aggregated pickle file into separate folders based on GPU model, using the GPU-node identifier (e.g., gpua for A100 nodes and gh for GH200 nodes).
(6) Compute intra-GPU error propagation probabilities and propagation times, using a maximum propagation window of 5 seconds, per each day.
(7) Compute inter-GPU error propagation probabilities and propagation times, using a maximum propagation window of 5 seconds, per each day.
(8) Aggregate the per-day intra-GPU propagation data into a single file.
(9) Aggregate the per-day inter-GPU propagation data into a single file.

Artifact Analysis (incl. Outputs)

The command generates the following key pickle files for each GPU-node model, saved in separate subdirectories. For instance, files corresponding to nodes labeled as gpua in the system will be located under the path `<output_dir>/gpua/`. The expected output files include:

(1) `_allfileparseddata_.pkl`: Contains a list of matched and parsed log lines, including metadata such as timestamp, datetime, hostname, process information, and the original log message.
(2) `*\_allfileregexbookkeeping\_*.pkl`: Stores the matched regular expression patterns and corresponding XID codes. Each entry aligns with the records in `_allfileparseddata_.pkl` one-to-one.
(3) `*_same_dev_id_same_node_*.pkl`: Captures intra-GPU error propagation probabilities and propagation time distributions for each node.
(4) `*_same_node_diff_dev_id_*.pkl`: Captures inter-GPU error propagation probabilities and propagation time distributions for devices within the same node.

The count statistics generated from item (1) and (2) per GPU XID error types are expected to match the counts in Table 1, and the corresponding system-wide and per-node MTBEs are then computed by the method specified in the paper's methodology section (Section 3.2). Moreover, items (3) and (4) generate the propagation figures (Figures 5-7) for GPU memory, GPU hardware, and NVLink errors. The error propagation probability and propagation time are extracted by aggregating all errors of the same XID and should match those presented in the error propagation figures. Because of noise in error propagation analysis, we omit the edges for which the propagation probability is less than 1%.

C.2 Computational Artifact A_2

Artifact Setup (incl. Inputs)

Artifact A_2 (*JobDetailCollect.sh* and *ApplicationStatsPipeline.ipynb*) collects user job metadata using `sacct` and analyzes the impact of GPU errors on user job resilience.

To prepare the runtime environment for executing A_2:

(1) `JobDetailCollect.sh` uses the `sacct` command, which is specific to systems running the `Slurm` workload manager. We include `JobDetailCollect.sh` for completeness. Systems

not using `Slurm` should instead employ equivalent tools to collect job metadata as described below, in a manner analogous to using `sacct`.
(2) `ApplicationStatsPipeline.ipynb` can be executed in the same software and hardware environment prepared for A_1.

The inputs to Artifact A_2 are as follows:

(1) `JobDetailCollect.sh` requires a `bash` environment and access to the `sacct` command to collect user job information.
(2) `ApplicationStatsPipeline.ipynb` requires user job metadata extracted by `JobDetailCollect.sh` and the following pickles generated by Artifact A_1: `*_allfileparseddata_*.pkl` and `*_allfileregexbookkeeping_*.pkl`.

For systems not using `sacct`, equivalent tools may be used to collect the following user job metadata from `Slurm`:

```
user,jobid,jobname,partition,state,exitcode,start,end,elapsed,
     nodelist,alloctres
```

Here, `nodelist` specifies the list of nodes on which the job was scheduled, and `alloctres` indicates the resources allocated to the job. To protect user privacy and sensitive information, we are not allowed to release the original user job records from *Delta*. However, we believe that Artifact A_2 is broadly applicable to other systems with comparable job metadata and workload management capabilities.

Artifact Execution

Run the following commands to execute A_2:

```
chmod +x ./JobDetailCollect.sh

./JobDetailCollect.sh
```

for collecting user job metadata using `sacct`;

```
jupyter notebook ./ApplicationStatsPipeline.ipynb
```

for launching the application statistic analysis Jupyter notebook.

Once the notebook is launched, set the `parsed_logs_file` and `regex_template_file` variables to point to `allfileparseddata.pkl` and `allfileregexbookkeeping.pkl` pickle files generated by Artifact A_1, respectively. Executing each cell in the Jupyter notebook performs the following analysis steps:

(1) Classifies job failures and associates them with GPU XID errors, using both the collected job metadata and GPU error records contained in the A_1 generated pickle files.
(2) Produces a statistical summary of the jobs, including GPU count, runtime duration, failure rates, and estimated GPU hours for ML and non-ML workloads, based on the collected job metadata.
(3) Estimates node drain and recovery times by analyzing inter-arrival times between jobs on each GPU node subject to GPU errors.

Artifact Analysis (incl. Outputs)

The expected outputs of this artifact are:

(1) TXT files containing job metadata collected using `sacct`, capturing job resource usage, runtime characteristics, and execution state.
(2) JSON files containing classified job records, labeled as GPU-failed or successfully completed, based on temporal correlation with GPU error logs.
(3) A summary table of job statistics, including GPU count, runtime (mean and P99), failure rates, and estimated GPU-hours for ML and non-ML workloads.
(4) Bar-plot visualizations for A100 and H100 GPUs showing the distribution of node drain durations and recovery times.

The classified job failure records generated by item (2) are expected to match the GPU error-induced failure statistics shown in Table 2. The aggregate job metrics computed in item (3) should match the job usage breakdowns and failure distributions in Table 3. The node unavailability times derived in item (4) should reproduce the distributions shown in Figure 10 for both A100 and H100 GPUs.

C.3 Computational Artifact A_3

Artifact Setup (incl. Inputs)

Artifact A_3 (*DCGMUtilizationAnalysis.py*) analyzes GPU utilization behavior before and after a GPU XID error by processing data recorded by NVIDIA's Data Center GPU Manager (DCGM), collected from *Delta*. Specifically, it examines the `DCGM_FI_DEV_GPU_UTIL` field, which tracks GPU utilization over time.

The runtime environment requirements for executing A_3 are as follows:

(1) `Python` 3.12.7
(2) `matplotlib` version 3.9.2
(3) `numpy` version 1.26.4
(4) `pandas` version 2.2.2

The inputs required by A_3 are:

(1) DCGM data collected in CSV format. The file must contain the following columns for analysis: `time`, `gpu_id`, `host`, and `DCGM_FI_DEV_GPU_UTIL`. The `DCGM_FI_DEV_GPU_UTIL` field is the identifier for GPU utilization in DCGM.
(2) GPU failure incident metadata provided as manual inputs: *incident date, host identifier, GPU ID, DCGM data start date, and DCGM data end date.*

Artifact Execution

To run Artifact A_3, modify the `cases` variable to specify the desired `incident date`, `host identifier`, `GPU ID`, `DCGM data start date`, and `DCGM data end date`. Additionally, set the `DATA_PATH` variable to point to the directory containing the relevant DCGM CSV record files. Then, execute the following command:

```
python3 DCGMUtilizationAnalysis.py
```

The filenames of the DCGM CSV files must follow the format:

```
<datadir>/daily/{GPU_TYPE}data_from{file_start_date}to{file_end_date
     }.csv
```

Here, `GPU_TYPE` is a macro defined within the script, and `file_start_date` and `file_end_date` must follow the `yyyy-mm-dd` date format.

A_3 extracts GPU utilization data from the `DCGM_FI_DEV_GPU_UTIL` field in the DCGM dataset, focusing on a 10-minute window surrounding the user-provided GPU error incident (identified by Artifact A_1). It then visualizes the utilization characteristics using line plots.

Artifact Analysis (incl. Outputs)

The output data generated from this artifact are line-plots that are similar to those in Figure 9 of the paper.

C.4 Computational Artifact A_4

Artifact Setup (incl. Inputs)

Artifact A_4 (*OverprovisioningEmulation.py*) emulates the availability of a large-scale GPU system by modeling the effects of random node failures, average node downtime, total job runtime, and varying levels of overprovisioning.

A_4 can be executed using the same software environment as Artifact A_3. It does not require any input files; all system parameters can be configured by modifying the system parameters constants defined at the top of the script.

Artifact Execution

Artifact A_4 emulates system availability to determine the optimal overprovisioning cost—defined as the number of spare GPUs—required to achieve a specified availability threshold. Specifically, this script:

(1) Generates a list of unavailability intervals by simulating GPU node failures using the failure rate derived from the MTBE statistics computed by Artifact A_1. Each failure interval has a duration equal to the average GPU downtime estimated by Artifact A_2, and multiple GPU failure intervals may overlap.
(2) For a given overprovisioning level (i.e., the number of spare GPUs), calculates the total duration during which the number of simultaneously failed GPUs exceeds the available spare capacity, thereby violating the overprovisioning limit. The total duration refers to the cumulative system downtime as perceived from the user job's perspective.
(3) Repeats step (2) across varying overprovisioning cost levels to identify the minimal number of spare GPUs required to maintain availability above the specified threshold.

To execute Artifact A_4, run the following command:

```
python3 OverprovisioningEmulation.py
```

Artifact Analysis (incl. Outputs)

Artifact A_4 outputs the optimal overprovisioning cost based on user-defined system resilience parameters and the specified availability threshold. When configured with parameters consistent with those presented in the paper, the artifact produces results that align with the overprovisioning costs reported in Section 5.